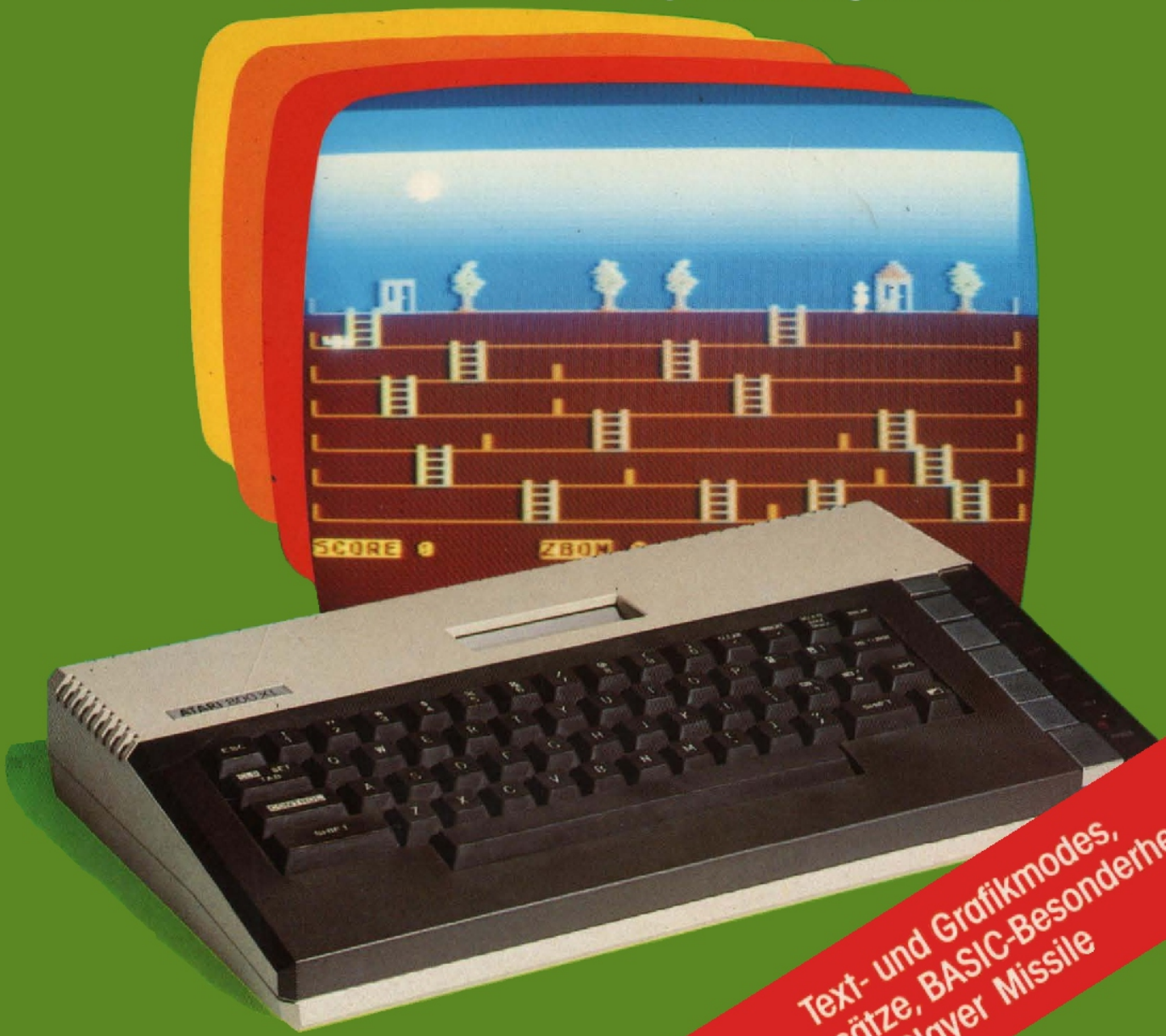
HAPPY COMPUTER

Hans Lorenz Schneider
Rudolf Bichler

Das ATARI-Buch

Band 1: Grundlegende Programmiermöglichkeiten



Text- und Grafikmodes, Zeichensätze, BASIC-Besonderheiten, Player Missile

Ein Markt & Technik Buch

Das ATARI-Buch

Hans Lorenz Schneider
Rudolf Bichler

Das ATARI-Buch

Band 1: Grundlegende Programmiermöglichkeiten

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schneider, Hans Lorenz:
Das ATARI-Buch / Hans Lorenz Schneider ; Rudolf Bichler. —
Haar bei München : Markt-u.-Technik-Verlag
(Happy computer)
NE: Bichler, Rudolf:
Bd. 1. Grundlegende Programmiermöglichkeiten. — 1984.
ISBN 3-89090-039-9

Die Informationen im vorliegenden Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Buch gezeigten Modelle und Arbeiten ist nicht zulässig.

ATARI® ist ein Warenzeichen der Atari, Inc., USA
(eine Warner Communications Company)

15 14 13 12 11 10 9 8 7 6 5 4 3 2

88 87 86 85

ISBN 3-89090-039-9

© 1984 by Markt & Technik, 8013 Haar bei München
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

Vorwort

Wie der Titel schon sagt, möchten wir Ihnen mit diesem Buch grundlegende Programmiermöglichkeiten Ihres ATARI-Computers aufzeigen. Dieses Buch beinhaltet keinen Basic-Kurs, dazu sei auf die entsprechende Literatur verwiesen (z.B. die Buchreihe 'Basic ohne Probleme'). Weiterhin wird die Kenntnis des ATARI-Handbuches vorausgesetzt. Trotzdem haben wir versucht Ihnen auf einfache Art und Weise die doch recht komplizierten Vorgänge in Ihrem Computer darzulegen.

Benutzen Sie das erste Kapitel, um sich etwas mit Ihrem Rechner vertraut zu machen. Es ist kein Hindernis, wenn Sie den mathematischen Hintergrund zu diesem Programm nicht voll und ganz verstehen.

Von Hause aus sind die ATARI-Computer als Spielcomputer konzipiert. Dies wirkt sich auch in ihrer Handhabung bei der Programmierarbeit aus. Effektvolle Spiele sind relativ leicht zu programmieren, aber die gesamte Handhabung des ATARI wird dadurch etwas komplizierter. Besonderen Wert haben wir auf die verschiedenen Möglichkeiten der Grafik- und Textdarstellung gelegt. Auch eine genaue Beschreibung der Player Missile wurde in diesen Band einbezogen.

Da in dem Buch eine dichtgedrängte Informationsfülle vorliegt, wird sich der Computerneuling zu Beginn etwas schwer tun. Aber auch ihm hilft hier das alte Computerprinzip: TRY AND ERROR.

Auch bei diesem Buch würden wir uns wieder über konstruktive Kritik der Leser freuen. Und nun wünschen wir Ihnen viel Spass beim Programmieren und Ausprobieren!

Hans Lorenz Schneider **Rudolf Bichler**

Einleitung

Mit diesem Buch haben wir versucht Ihnen die grundlegenden Eigenschaften der ATARI-Computer 600 und 800 XL zu vermitteln. Größtenteils sind die dargestellten Programme und Programmteile auch auf den Rechnern der älteren Serie lauffähig und in Einzelfällen haben wir auf Unterschiede hingewiesen.

Das erste kapitel beschäftigt sich mit einem mathematischen Spiel - für Sie zur Eingewöhnung. Um das Spiel ganz zu verstehen, müßte man tief in die Mathematik einsteigen. Wir haben dieses Beispiel deshalb ausgesucht, weil es typisch für die Arbeitsweise eines Rechners ist, außerdem dürfte dieses Spiel auch vielen Lesern als Streichholzspiel bekannt sein.

Das zweite Kapitel beschäftigt sich mit den verschiedenen Ausgabemöglichkeiten Ihres ATARI-Computers am Bildschirm: Text und Grafik. Zunächst gehen wir dabei auf die diversen Zeichensätze und Textmodes ein. Da der ATARI-Computer hauptsächlich für Spiele konzipiert ist, stehen hier einige Möglichkeiten zur Verfügung. Noch ein paar Möglichkeiten mehr gibt es bei den Grafiksmodes, die wir in Kapitel 2.2 behandeln. Das letzte Kapitel dieser Reihe beschäftigt sich mit dem Mischen von Texten und Grafiken, wobei im Besonderen die - dazu sehr wichtige - Display List eingehend erläutert wird.

Kein Home-Computer kommt heute mehr ohne Figuren aus, die als Ganzes, per einfachem Befehl, zu bewegen sind. Was bei anderen Computern die Sprites sind, sind bei dem ATARI-Computer die Player Missile. Wir gehen näher ein auf ihre Definition, ihre Bewegung, die Prioritäten untereinander und gegenüber dem Hintergrund und Kollisionen zwischen Player, Missiles und dem Hintergrund.

Kapitel 4 ist für diejenigen gedacht, die Basic schon genügend beherrschen und sich nun für die Unterschiede zu anderen Computern interessieren. In diesem Kapitel werden einige Besonderheiten des ATARI-Basic besprochen, wie z.B. das Speichern von Maschinenroutinen, der XIO-Befehl und die String-Initialisierung. Vervollständigt wird das Ganze durch ein Programm zum Löschen von Zeilen, einer Übersicht und Handhabungshilfe für Tabellen und Zeigern, sowie zwei Programmen, die Zeilen umnumerieren (ohne Berücksichtigung

von Sprüngen) und deutsche Fehlermeldungen, statt der beim ATARI üblichen Nummern produzieren.

Der Anhang soll Ihnen hauptsächlich als Nachschlagewerk dienen. Während des gesamten Buches wurde ein Beispiel immer wieder ausgebaut. Dieses Beispiel werden wir in den weiteren Bänden auch ständig verbessern. Um Ihnen das mühsame Zusammensuchen der einzelnen Programmteile aus dem Buch zu ersparen, haben wir im ersten Teil des Anhangs ein Gesamtlisting unseres Spieles 'BERKWERK' abgebildet. Darauf folgt eine Übersicht über die verwendeten Variablen und eine Tabelle der aufgerufenen Unterprogramme.

Da der verwendete Drucker nicht alle Zeichen (Kleinerzeichen, Größereichen) darstellen kann, wurden diese in spezifischer Weise im Text abgekürzt. Eine Übericht dieser Abkürzungen finden Sie in dem Teil Parametertypen. Weiterhin sind hier die verschiedenen Variablentypen aufgelistet, wie sie in den Variablenübersichten verwendet werden.

Anhang 4 bringt eine Übersicht über den internen Code und den ATASCII-Code jedes einzelnen Zeichens. Für den geübten Programmierer sicherlich eine sehr gute Hilfe.

Die nächsten beiden Teile sind den Grafikmodes gewidmet. Anhang 5 bietet hier eine Übersicht über alle Grafikmodes und ihre Ansteuerung und Anhang 6 ist speziell für die Besitzer der 400/800er-Modelle.

Wer seinen ATARI voll ausnutzen möchte, braucht bestimmte Informationen über das Betriebssystem. Hier sind besonders wieder die Register, die vom Anwender geändert werden können, wichtig. Eine Übersicht über diese Betriebssystem-Variablen geben wir in Anhang 7 wieder.

Es folgen zwei Assemblerlistings für im Buch verwendete Maschinenprogramme, die dem Assemblerprogrammierer deren Wirkungsweise besser verdeutlichen.

Den Abschluß bildet ein Inhaltsverzeichnis der lieferbaren Disketten. Die Diskette bieten wir allen denjenigen an, denen die reine Tipparbeit zuviel Arbeit ist, und die die vorhandenen Programme ausbauen, ergänzen oder verbessern wollen.

Das ATARI - Buch

Band 1: Grundlegende Programmiermöglichkeiten

Inhaltsverzeichnis

Vorwort 5

Einleitung 7

Inhaltsverzeichnis 9

1. NIM - Ein Spiel zum Eingewöhnen 11

2. Text und Grafik 21

2.1 Zeichensätze und Textmodes 25

2.2 Grafikmodes 62

2.3 Texte und Grafiken mischen (Display List) 66

3. Player Missiles 79

3.1 Definition 81

3.2 Bewegung 88

3.3 Prioritäten 90

3.4 Kollision 91

4. Basic-Besonderheiten 95

4.1 Maschinenroutinen speichern 97

4.2 XIO-Befehl 100

4.3 String-Initialisierung 101

4.4 Zeilen löschen 102

4.5 Tabellen und Zeiger 103

4.6 RENUMBER / Fehlermeldungen 114

Anhang 125

Anhang 1: Gesamtlisting 'BERGWERK' 127

Anhang 2: Variablenübersicht 'BERGWERK' 133

Anhang 3: Parametertypen 135

Anhang 4: Interner Code und ATASCII-Code der Zeichen 136

Anhang 5: Grafikmodes 142

Anhang 6: GRAPHICS 12 für 400/800er-Besitzer 144

Anhang 7: Betriebssystemvariablen 145

Anhang 8: Assemblerlisting Zeichensatz kopieren 155

Anhang 9: Assemblerlisting Buchstaben mit Unterlängen 156

Anhang 10: Inhaltsverzeichnis der Diskette 158

1

NIM — Ein Spiel zum Eingewöhnen

1. NIM - Ein Spiel zum Eingewöhnen

Das NIM-Spiel wurde bewußt an den Anfang dieses Buches gestellt, weil es zum einen ein einfaches und interessantes Spiel ist, und zum anderen eine gute Übungsmöglichkeit für das Rechnen mit Binär-Zahlen (Dual-Zahlen) ist (Dual-Zahlen spielen eine grundlegende Rolle in der Arbeit mit Rechnern). Obwohl die mathematische Lösung erklärt wird, sollte man dieses Spiel erst einmal 'intuitiv', d.h. durch reines Nachdenken spielen, und man wird sehen, daß der Computer fast alle Spiele gewinnt.

Übungsziel:

- Einführung in das Atari-Basic
- Dualzahlen

Spielanleitung

NIM ist ein Zwei-Personen-Spiel, bei dem abwechselnd Streichhölzer aus einer Grundmenge entfernt werden. Dabei liegen die Streichhölzer in mehreren Reihen, und es darf pro Zug eine beliebige Anzahl in **einer** Reihe entfernt werden.

Wer das letzte Streichholz nimmt, hat gewonnen.

Diese Spielanleitung wird im Programm wiederholt.

Spielstrategie

Bevor wir näher auf das Programm eingehen, wollen wir uns zunächst einmal etwas mit der Spielstrategie befassen.

In jeder Spielsituation kann eindeutig festgestellt werden, ob es sich um eine Gewinn- oder Verlustsituation (für denjenigen der nicht am Zug ist) handelt.

Eine Gewinnsituation wird durch jeden Zug zu einer Verlustsituation, aber eine Verlustsituation kann sowohl in eine Gewinn- als auch wieder in eine Verlustsituation

übergeführt werden. D.h. gelingt einem Spieler eine Gewinnsituation herbeizuführen, so muß der andere Spieler einen Zug machen, der zu einer Verlustsituation führt. Dadurch kann aber der erste Spieler wieder eine Gewinnsituation herbeiführen, wenn er seinen Zug entsprechend wählt. Bei optimalem Spiel wird also derjenige Spieler gewinnen, der als erster eine Gewinnsituation herbeiführt und sie anschließend nicht mehr abgibt.

Dadurch ergeben sich zwei Probleme:

1. Wie erkenne ich eine Gewinnsituation ?

2. Wie finde ich in einer Verlustsituation den Zug, der zu einer Gewinnsituation führt ?

Erkennen einer Gewinnsituation

Um den Gewinn- bzw. Verlustcharakter einer Situation zu erkennen, muß man die Anzahl der Hölzchen jeder Reihe in Dualzahlen (Potenzen der Zahl 2) zerlegen. Von diesen Dualzahlen addiert man jeweils die einzelnen Stellen getrennt voneinander. Ist **jede** Summe, die man dabei erhält gerade, so liegt eine Gewinnsituation vor, d.h. für den Spieler, der am Zug ist eine Verlustsituation.

Beispiel

Reihe	Anzahl	Dualzahl
1	1	001
2	3	011
3	5	101
4	7	111
	Summe	224 : Gewinnsituation

Bestimmen eines Zuges um eine Verlustsituation in eine Gewinnsituation umzuwandeln

Schreibt man nun für jede Summe eine Null, wenn die Summe gerade ist, bzw. eine Eins, wenn die Summe ungerade ist, so erhält man, wenn man die Nullen und Einsen gemäß ihrer Stelle hintereinander schreibt, wieder eine Zahl, die wir Situationszahl nennen wollen. Eine Gewinnsituation liegt also genau dann vor, wenn die Situationszahl Null ist. Um nun eine Verlustsituation in eine Gewinnsituation zu verwandeln, muß man eine Zahl (Anzahl der Hölzchen in einer

Reihe) finden, die an der höchsten Stelle der Situationszahl eine Eins hat, wie man durch Nachrechnen einiger Beispiele leicht feststellen kann. Zwischen dieser Zahl und der Situationszahl muß dann eine logische EXCLUSIV ODER-Verknüpfung durchgeführt werden. Die so erhaltene Zahl ist dann die neue Anzahl in dieser Reihe, und damit liegt eine Gewinnsituation vor.

```
Beispiel:  1. Reihe  1  001
           2. Reihe  5  101
           3. Reihe  7  111
                        ---
      Situationszahl    011

         111 EOR 011 = 100 (4)    (EOR - Exclusiv ODER)
      und 1,4,5 ist Gewinnsituation.
```

Programmlisting

```
1000 REM ***************************************************48
1010 REM *                  NIM SPIEL                      *
1020 REM ***************************************************48
1030 REM
1040 REM
1050 REM ***************************************************48
1060 REM *                   ANLEITUNG                     *
1070 REM ***************************************************48
1080 GRAPHICS 0
1090 SETCOLOR 4,1,2
1100 SETCOLOR 0,1,10
1110 SETCOLOR 2,1,2
1120 POKE 752,1:REM ------------ CURSOR AUS -------------
1130 PRINT "        ##  ##  ##  ##   ##"
1140 PRINT "        ### ##  ##  ### ###"
1150 PRINT "        ######  ##  #######"
1160 PRINT "        ######  ##  ## # ##"
1170 PRINT "        ## ###  ##  ##   ##"
1180 PRINT "        ##  ##  ##  ##   ##"
1190 PRINT :PRINT :PRINT
1200 PRINT "NIM IST EIN ZWEIPERSONENSPIEL, WO AB-"
1210 PRINT "WECHSELND STREICHHOELZER AUS EINER"
1220 PRINT "GRUNDMENGE ENTFERNT WERDEN."
1230 PRINT "DABEI LIEGEN DIE STREICHHOELZER IN"
1240 PRINT "MEHRERENREIHEN UND ES DARF PRO ZUG"
1250 PRINT "EINE BELIEBIGE ANZAHL IN  E I N E R"
1260 PRINT "REIHE ENTFERNT WERDEN."
1270 PRINT "WER DAS LETZTE WEGNIMMT HAT GEWONNEN."
1280 PRINT
1290 PRINT "UND NUN VIEL SPASS (TASTE DRUECKEN)"
```

```
1300 OPEN #1,4,0,"K:"
1310 GET #1,TASTE
2000 REM ***********************************************
2010 REM *                    VORSPANN                 *
2020 REM ***********************************************
2030 GRAPHICS 2
2040 SETCOLOR 2,0,0
2050 POKE 752,1
2060 PRINT "{WIEVIELE ZEILEN (2-9) ";:INPUT ANZZEI
2070 IF ANZZEI<2 OR ANZZEI>9 THEN 2060
2080 PRINT "{WIEVIELE HOELZCHEN MAXIMAL (2-15) ";
2090 INPUT ANZHOE
2100 IF ANZHOE<2 OR ANZHOE>15 THEN 2080
2110 DIM ANZAHL(ANZZEI)
2120 FOR ZEILE=1 TO ANZZEI
2130 ANZAHL(ZEILE)=INT(ANZHOE*RND(0))+1
2140 NEXT ZEILE
2150 FOR ZEILE=1 TO ANZZEI
2160 GOSUB 7000
2170 NEXT ZEILE
2180 PRINT "{WILLST DU BEGINNEN (J/N) ? "
2190 GET #1,TASTE
2200 IF TASTE=ASC("J") THEN 3000
2210 IF TASTE=ASC("N") THEN 4000
2220 GOTO 2190
3000 REM ***********************************************
3010 REM *                  SPIELERZUG                 *
3020 REM ***********************************************
3030 PRINT "{"
3040 TRAP 3040:PRINT "GIB GEWUENSCHTE REIHE AN";
3050 INPUT ZEILE1
3060 IF ZEILE1<1 OR ZEILE1>ANZZEI THEN 3040
3070 PRINT "{WIEVIELE NIMMST DU WEG";:INPUT WEG
3080 WEG=INT(WEG)
3090 IF WEG<1 OR WEG>ANZAHL(ZEILE1) THEN 3070
3100 ANZAHL(ZEILE1)=ANZAHL(ZEILE1)-WEG
3110 GOSUB 8000
3120 IF ANZZEI>0 THEN 4000
3130 GRAPHICS 2:POSITION 0,3:PRINT #6;"     GRATULIERE"
3140 PRINT #6:PRINT #6;"  DU HAST GEWONNEN"
3150 FOR SCHLEIFE=1 TO 1000:NEXT SCHLEIFE
3160 RUN
4000 REM ***********************************************
4010 REM *                   COMPUTERZUG               *
4020 REM ***********************************************
4030 GOSUB 9000
4040 IF SIT=0 THEN 4500
4050 REM ------------- RICHTIGEN ZUG SUCHEN -------------
4060 POT=1
4070 IF POT>SIT THEN 4090
4080 POT=2*POT:GOTO 4070
```

```
4090 POT=POT/2
4100 ZEILE1=0
4110 ZEILE1=ZEILE1+1
4120 VAR=INT(ANZAHL(ZEILE1)/POT)/2
4130 IF VAR=INT(VAR) THEN 4110
4140 VAR=ANZAHL(ZEILE1)
4150 ANZAHL(ZEILE1)=0
4160 GOSUB 9000
4170 ANZAHL(ZEILE1)=SIT
4180 GOSUB 8000
4190 IF ANZZEI>0 THEN 4220
4200 GRAPHICS 2:PRINT "TJA - ICH HABE GEWONNEN."
4210 GOTO 3150
4220 PRINT "↖ICH HABE IN DER ";ZEILE1;". ZEILE "
4230 IF SIT=0 THEN PRINT "ALLE";:GOTO 4250
4240 PRINT VAR-SIT;
4250 PRINT " HOELZCHEN GENOMMEN."
4260 GOTO 3040
4500 REM ------------------ ZUFALLSZUG ------------------
4510 ZEILE1=INT(RND(0)*ANZZEI)+1
4520 VAR=ANZAHL(ZEILE1)
4530 ANZAHL(ZEILE1)=INT(RND(0)*ANZAHL(ZEILE1))
4540 SIT=ANZAHL(ZEILE1)
4550 GOSUB 8000
4560 GOTO 4220
7000 REM ***************************************************
7010 REM *             SPIELSITUATION ZEICHNEN             *
7020 REM ***************************************************
7030 FOR SCHLEIFE=1 TO 50
7040 REM ------------------ WARTESCHLEIFE ---------------
7050 NEXT SCHLEIFE
7060 POSITION 0,ZEILE-1:? #6;ZEILE;
7070 FOR SCHLEIFE=1 TO (17-ANZAHL(ZEILE))/2
7080 PRINT #6;" ";
7090 NEXT SCHLEIFE
7100 FOR SCHLEIFE=1 TO ANZAHL(ZEILE)
7110 PRINT #6;"¡";
7120 NEXT SCHLEIFE
7130 FOR SCHLEIFE=1 TO (18-ANZAHL(ZEILE))/2
7140 PRINT #6;" ";
7150 NEXT SCHLEIFE
7160 IF ANZAHL(ZEILE)<10 THEN PRINT #6;" ";
7170 PRINT #6;ANZAHL(ZEILE);
7180 RETURN
8000 REM ***************************************************
8010 REM *             SPIELFELD AKTUALISIEREN             *
8020 REM ***************************************************
8030 IF ANZAHL(ZEILE1)=0 THEN 8050
8040 ZEILE=ZEILE1:GOSUB 7000:RETURN
8050 IF ZEILE1=ANZZEI THEN 8090
8060 FOR SCHLEIFE=ZEILE1 TO ANZZEI-1
```

```
8070 ANZAHL(SCHLEIFE)=ANZAHL(SCHLEIFE+1)
8080 NEXT SCHLEIFE
8090 ANZZEI=ANZZEI-1
8100 FOR ZEILE=1 TO ANZZEI
8110 GOSUB 7000
8120 NEXT ZEILE
8130 POSITION 0,ANZZEI:? #6;"                    "
8140 RETURN
9000 REM ***************************************************
9010 REM *        TEST AUF GEWINN/VERLUST-STELLUNG        *
9020 REM ***************************************************
9030 SIT=0:POT=1
9040 SUMME=0
9050 FOR ZEILE=1 TO ANZZEI
9060 SUMME=SUMME+INT(ANZAHL(ZEILE)/POT)
9070 NEXT ZEILE
9080 IF SUMME=0 THEN RETURN
9090 POT=2*POT:SIT=SIT+POT*(SUMME/2-INT(SUMME/2))
9100 GOTO 9040
```

Programmbeschreibung

Aufteilung des Programmes nach Zeilennummernbereichen:

Hauptprogramm

1000 Anleitung
2000 Vorspann
3000 Spielerzug
4000 Computerzug

Unterprogramme

7000 Spielsituation zeichnen
8000 Spielfeld aktualisieren
9000 Test auf Gewinn-/Verluststellung

Die Unterprogramme von 7000 bis 7180 und von 8000 bis 8140 dienen zum Zeichnen einer Reihe, wobei vor jeder Reihe die Nummer jeder Reihe geschrieben wird (Zeile 7060), und hinter jeder Reihe die Anzahl der Hölzchen in dieser Reihe (Zeilen 7160 und 7170). In den Zeilen 8050 bis 8130 werden die Zeilen zusammengerückt, falls dies durch leer werden einer Zeile nötig geworden ist. Die Darstellung der Hölzchen durch 'I's, ist nicht sehr schön, und wir werden deshalb im zweiten Kapitel dieses Buches kennenlernen, wie man sich einen neuen Zeichensatz definiert.

Im Unterprogramm von 9000 bis 9100 wird die Situationszahl

SIT errechnet, wobei wir jedoch auf eine Zerlegung in Dualzahlen verzichten können, wie das folgende Beispiel zeigt. (Die Rechnung ist nicht anders, nur leichter zu handhaben)

Reihe	Anzahl	INT(Anzahl/2)	INT(Anzahl/4)
1	1	0	0
2	2	1	0
3	2	1	0
Summe	5	2	0
ergibt	1	0	0

SIT = 001, d.h. Verlustsituation

Ist die Situationszahl gleich Null (Abfrage in 4040), so wird in den Zeilen 4500 bis 4560 zufällig ein Hölzchen weggenommen. Bei positiver Situationszahl, d.h. einer Verlustsituation wird in den Zeilen 4060 bis 4090 die höchste Stelle dieser Situationszahl errechnet. In den Zeilen 4100 bis 4130 wird dann eine Zeile gesucht, in der an dieser Stelle eine Eins steht. Da eine logische EXCLUSIV ODER-Verknüpfung zweier Zahlen in Basic relativ kompliziert zu realisieren ist, verwenden wir, um die neue Anzahl in dieser Zeile zu finden, einen kleinen Trick. Wir setzen die Anzahl in dieser Zeile gleich Null (Zeile 4150) und springen dann in das Unterprogramm in Zeile 9000. Die errechnete Situationszahl ist dann die Anzahl in dieser Zeile. Der errechnete Spielzug wird dann in den Zeilen 4190 bis 4250 ausgegeben. Da in dem Unterprogramm von 8000 bis 8140 leere Zeilen immer aufgefüllt werden, ist der Test auf Gewinn oder Verlust leicht zu realisieren. Das Spiel ist nämlich genau dann beendet, wenn die Anzahl der Zeilen gleich Null ist.

Wer sich näher mit diesem oder ähnlichen Spielen befassen will, sei auf Bücher über Spieltheorie verwiesen.

Variablenübersicht

Da wir bei allen Programmen eine Variablenübersicht darstellen wollen, dient die folgende Übersicht (wegen der geringen Zahl von Variablen) weniger der Übersicht, als der Eingewöhnung.

NIM **1000 - 9100**

Variablen:

Name	Bedeutung
ANZHOE	Maximale Anzahl in einer Zeile (Eingabe)
ANZZEI	aktuelle Anzahl der Zeilen
POT	Zweierpotenz
SCHLEIFE	allgemeine Schleifenvariable
SIT	Situationszahl, bzw. neue Anzahl in der aktuellen Zeile
SUMME	dient zur Berechnung von SIT (s.Text)
TASTE	ATASCII Code der gedrückten Taste
VAR	Hilfsvariable
WEG	Anzahl der weggenommen Hölzchen
ZEILE	Hilfsvariable für Upgr. 7000-7180
ZEILE 1	aktuelle Zeile

Felder:

Name	Bedeutung
ANZAHL	aktuelle Spielsituation (Anzahl der Hölzchen pro Zeile) / 1...ANZZEI

Unterprogrammaufrufe:

in	nach	Zweck
2160	7000	Ausgabe Spielsituation
3110	8000	Spielfeld aktualisieren
4030	9000	Test Gewinn-/Verluststellung
4160	9000	Test Gewinn-/Verluststellung
4180	8000	Spielfeld aktualisieren
4550	8000	Spielfeld aktualisieren
8040	7000	Ausgabe Spielsituation
8110	7000	Ausgabe Spielsituation

2

Text und Grafik

2. Text und Grafik

Bevor wir näher auf die Besonderheiten des Atari-Computers eingehen, müssen wir uns mit der Speicherbelegung in unserem Computer etwas vertraut machen.

Speicherbelegung

Alle Atari-Computer haben einen Mikroprozessor vom Typ 6502 bzw. 65C02. Diese Mikroprozessoren können einen 64 KByte (= 65536) Byte großen Speicher verwalten. Eine Speicherzelle ist 8 Bit groß, d.h. sie kann Werte von 0 bis 255 annehmen.

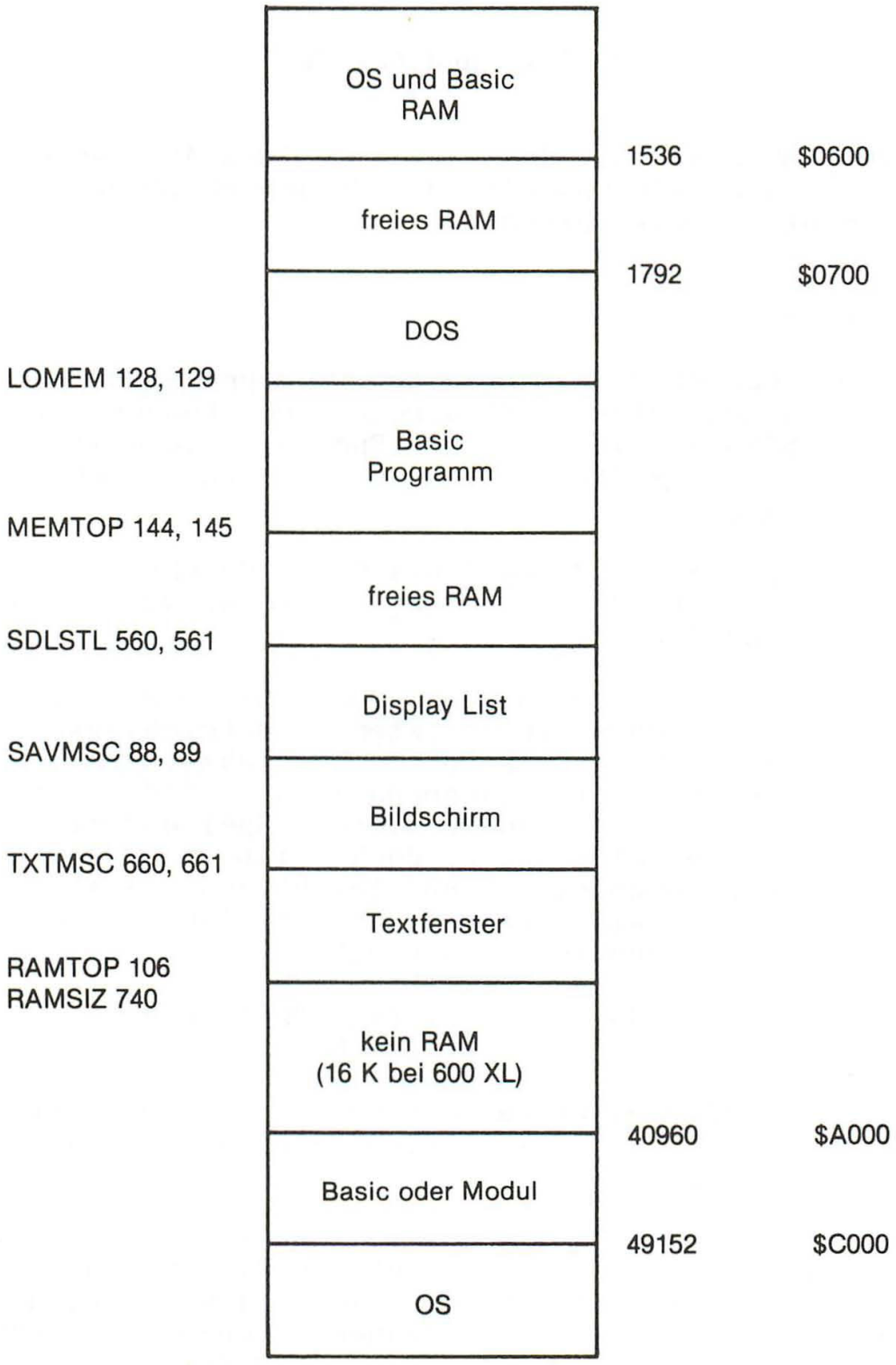
Die Speicherplätze sind von 0 bis 65535 durchnumeriert und der Zugriff erfolgt durch die Angabe dieser Nummer (Adresse des Speicherplatzes).

In der Tabelle unten erkennt man von 0 bis 1535 einen Bereich der OS-RAM (Operating System / Betriebssystem) benannt ist. Dieser Bereich wird vom Betriebssystem genutzt, um interne Variablen wie Cursorposition, Bildschirmfarben etc. zu speichern. Den Inhalt dieser Speicherzellen kann man durch POKE-Befehle ändern, doch sollte man damit vorsichtig umgehen, denn dies kann leicht zum 'Absturz' des Computers führen. Wir werden später einige dieser Speicherplätze für unsere Interessen nützen.

Der Bereich von 1536 bis 1791 ist frei, und wir werden dort kurze Maschinenprogramme speichern.

Bei Adresse 1792 beginnt das DOS (Disketten Operating System / Steuerung der Diskettenlaufwerke - sofern vorhanden).

Dahinter steht unser Basic-Programm (selbst wenn kein Basic-Programm im Computer ist, geht ab dieser Adresse ein Teil des Speichers verloren). In Kapitel vier werden wir uns diesen Speicherbereich genauer ansehen. Der Bereich hinter dem Basic-Programm ist für den Benutzer frei verfügbar, doch schwankt seine Größe je nach Speicherausbau, Länge des Basic-Programms und des verwendeten Grafikmodes. Deshalb sollte man bei Benutzung dieses Bereiches z.B. als Speicher für Daten vorsichtig sein, denn durch Vergrößerung des Basic-Programms können diese Daten verloren ge-


OS und Basic RAM
1536
$0600
freies RAM
1792
$0700
DOS
LOMEM 128, 129
Basic Programm
MEMTOP 144, 145
freies RAM
SDLSTL 560, 561
Display List
SAVMSC 88, 89
Bildschirm
TXTMSC 660, 661
Textfenster
RAMTOP 106
RAMSIZ 740
kein RAM (16 K bei 600 XL)
40960
$A000
Basic oder Modul
49152
$C000
OS

hen. Wir werden deshalb diesen Bereich nicht für unsere Daten benützen, sondern sie hinter den mit Bildschirm-RAM bezeichneten Bereich legen, wobei wir den Bildschirmspeicher nach hinten verschieben.

Auf die Speicherzellen ab Adresse 40190 bis zum Ende des Speichers haben wir in der Regel keinen Zugriff, denn dieser Bereich ist für den Basic-Interpreter (bzw. ein eingestecktes ROM-Modul) und das Betriebssystem reserviert. Zwar kann man auch diesen Bereich als RAM benützen, doch muß man sich dazu ein eigenes Betriebssystem schreiben, was nicht unbedingt jedermanns Sache ist.

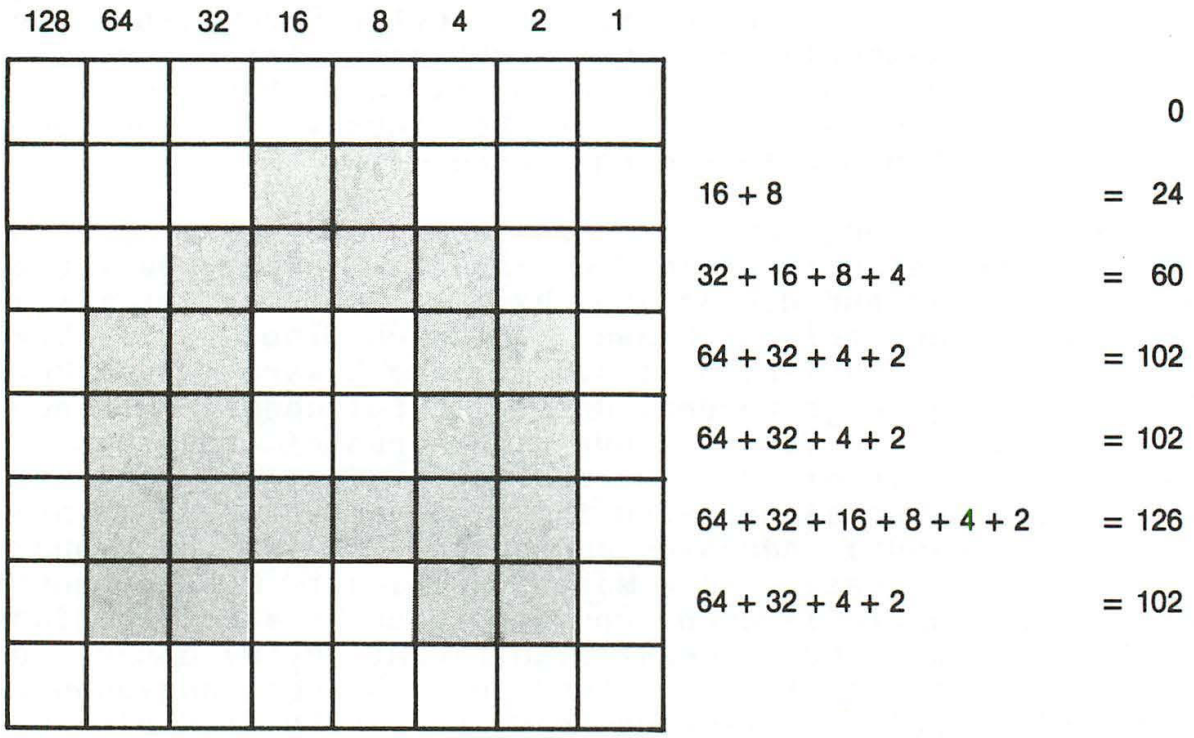
Da die oben angegebenen Bereiche nicht immer dieselbe Größe haben, sind links in der Tabelle jeweils Variablen angegeben, die auf den Anfang, bzw. das Ende eines jeweiligen Bereiches zeigen. Diese Variablen sind Basic- bzw. Betriebssystem Variablen, d.h. ihr Inhalt kann durch PEEK (Adresse) abgefragt werden. Dabei ist folgendes zu beachten: ist für eine Variable nur eine Adresse angegeben, so ist deren Inhalt mit 256 zu multiplizieren, um den richtigen Wert zu erhalten. Sind für eine Variable zwei (aufeinanderfolgende) Adressen angegeben, so ist der Inhalt der höheren Adresse (MSB = **M**ost **S**ignificant **B**yte genannt) mit 256 zu multiplizieren und dann der Inhalt der niedrigeren Adresse (LSB = Least **S**ignificant **B**yte) hinzuzuaddieren. Die in der Tabelle (und auch später) angegebenen Namen für Betriebssystem- und Basicvariablen sind diejenigen, die man auch in den Source Listings findet - sie spielen für die Programmierung aber selbstverständlich keine Rolle.

2.1 Zeichensätze und Textmodus

Wir betrachten uns zunächst einmal den normalen Textmode (GRAPHICS 0). In einem Zeichensatz sind die Punktmuster aller alphanumerischen und Sonderzeichen abgespeichert. Da beim Atari-Computer die Zeichen in einer 8x8 Matrix dargestellt werden, und man für jeden Punkt ein Bit (Punkt gesetzt, oder Punkt nicht gesetzt) benötigt, belegt die Codierung für ein Zeichen 8x8 = 64 Bit, d.h. 8 Byte. Dabei wird jeweils eine Reihe eines Zeichens zu einem Byte zusammengefaßt (siehe Zeichnung).

Es stehen insgesamt 256 Zeichen zur Verfügung, wobei jedoch im Zeichensatz nur 128 Zeichen codiert sind; die restlichen Zeichen sind die Inversen dieser 128 Zeichen. Da für die Codierung eines Zeichens 8 Byte benötigt werden, belegt somit den gesamten Zeichensatz 128x8 = 1024

Byte (genau 1 KByte). Der Standardzeichensatz des ATARI befindet sich ab Adresse 57344.



Die Betriebssystem-Variable CHBAS (Adresse 756) enthält einen Zeiger auf den Zeichensatz. Ruft man den Inhalt dieser Speicherzelle ab (PRINT PEEK (756)), so erhält man den Wert 224, der mit 256 multipliziert 57344 gibt. Wir können uns also einen eigenen Zeichensatz definieren, ihn in einen 'sicheren' RAM-Bereich (d.h. ein Speicherbereich der nicht durch ein Basic-Programm, oder durch einen Grafik-Modus etc. überschrieben werden kann) ablegen und dann den Zeiger CHBAS auf diesen RAM-Bereich stellen. Dabei ist zu beachten, daß der Wert in CHBAS eine durch vier teilbare Zahl sein muß, d.h. die Adresse des Zeichensatzes muß durch 1024 teilbar sein. Um nun einen 'sicheren' Speicherbereich für unseren eigenen Zeichensatz zu finden, betrachten wir nochmal die Tabelle der Speicherbelegung. Wie wir schon gesehen haben, ist der Platz zwischen BASIC und dem Bildschirmbereich für Datenspeicherung sehr kritisch. Also werden wir versuchen, den Bildschirmspeicher um 1024 Byte nach vorne (d.h. in Richtung Basic) zu verschieben, um so zwischen den Bildschirmspeicher und dem ROM einen

'sicheren' Speicherbereich zu erhalten.

Gibt man einen GRAPHICS-Befehl ein, so setzt das Basic, bzw. das Betriebssystem den Bildschirmspeicher an das oberste Ende des RAM-Bereichs. RAMTOP (Adresse 106) enthält einen Zeiger, der beim Einschalten und bei jedem RESET mit der Adresse des höchsten RAM-Bytes geladen wird. Da diese Adresse immer durch 256 teilbar ist, steht in RAMTOP nur das MSB. Also können wir uns mit der Befehlsfolge

```
POKE 106, PEEK (106)-8:GRAPHICS 0
```

einen Speicherbereich für unseren Zeichensatz reservieren. Wir ziehen 8*256 Byte ab, da beim Scrollen des Bildschirms noch 800 Bytes hinter dem Bildschirm-RAM gelöscht werden, und der Wert in RAMTOP durch 4 teilbar sein muß. Bei der Verwendung eines Graphicmodes, der mehr als 4k Speicher benötigt, muß der Wert in RAMTOP durch 16 teilbar sein.

Das folgende kleine Programm gibt ein Beispiel für die Neudefinition des Buchstabens A.

```
1000 POKE 106,PEEK(106)-4:GRAPHICS 0
1010 FOR I=0 TO 1023
1020 POKE 256*PEEK(106)+I,PEEK(57344+I)
1030 NEXT I
1040 A=256*PEEK(106)+33*8
1050 FOR I=A TO A+7:READ C:POKE I,C:NEXT I
1060 DATA 0,96,120,110,102,118,222,198
1070 POKE 756,PEEK(106)
```

In Zeile 1000 wird der Platz für den Zeichensatz frei gemacht. In den Zeilen 1010 bis 1030 wird der alte Zeichensatz aus dem ROM an diese Stelle kopiert. In den restlichen Zeilen wird schließlich der Buchstabe A verändert. Der Buchstabe A ist dabei das 33. Zeichen in dieser Tabelle, da sich die Reihenfolge der Zeichen im Zeichensatz nach dem internen Code (siehe Anhang) richtet.

In den Textmodes 1 und 2 gilt das eben Gesagte ebenso, mit dem einen Unterschied, daß nur die ersten 64 Zeichen (siehe Tabelle interner Code) dargestellt werden können. Alle anderen Zeichen werden als eines dieser ersten 64 Zeichen dargestellt, nur in einer anderen Farbe (siehe Tabelle Farbcodes). Also wird z.B. ein 'a' als 'A' in einer anderen Farbe dargestellt. Um die zweiten 64 Zeichen darzustellen, müssen wir den Wert in CHBAS um zwei erhöhen

(jetzt ist es nicht mehr nötig, daß der Wert durch 4 teilbar sein muß, es genügt Teilbarkeit durch 2).

Anders verhält es sich mit den Grafik-Modes 12 und 13. Hier ist ein Punkt eines Zeichens doppelt so breit wie ein Punkt in GRAPHICS 0. Ein Buchstabe ist demnach 4 x 8 Punkte groß. Da jeder Punkt vier Farbwerte annehmen kann (dafür werden 2 Bit benötigt), braucht man für die Codierung einer Zeile eines Buchstabens wieder ein Byte. Je nachdem, wie nun die beiden Bit, die einen Punkt bestimmen, gesetzt sind, wird die Farbe für diesen Punkt gewählt. Da bei vierfarbigen Buchstaben eine inverse Darstellung unsinnig ist, werden die inversen Zeichen genau wie die entsprechenden normalen Zeichen dargestellt, mit dem Unterschied, daß im Fall, wo für einen Punkt beide Bit gesetzt sind, eine andere Farbe gewählt wird (siehe Tabelle im Anhang).

Da es fast ein Ding der Unmöglichkeit ist, sich mit Papier und Bleistift einen fünffarbigen Zeichensatz herzustellen, stellen wir hier ein Hilfsprogramm zur Verfügung. Dieses Programm bietet nicht nur die Möglichkeit Zeichen auf einfache Art und Weise neu zu definieren, sondern ermöglicht auch das Zeichnen von ganzen Bildschirmen.

Dieses Programm erlaubt es, ohne weitere Hilfsmittel, einen neuen Zeichensatz zu definieren und diesen zu speichern. Außerdem kann mit dem Programm ein Bild gezeichnet werden, das ebenfalls sowohl auf Diskette wie auch auf Kassette für späteren Gebrauch gespeichert werden kann.

Bedienunganleitung

Startet man das Programm mit RUN, so gelangt man in das Hauptmenü. In diesem Menü hat man acht Möglichkeiten, das Programm fortzusetzen. Gehen wir die Optionen einzeln durch.

Farbe

Damit können wir alle fünf Farben verändern. Bewegt man den Joystick nach oben bzw. nach unten, so kann man ein Farbfeld auswählen, das sich durch Blinken bemerkbar macht. Bewegt man den Joystick nach rechts bzw. nach links, so ändert sich die Farbe in diesem Farbfeld. Dabei kommt es vor, daß sich die Hintergrundfarbe im Textfenster oder auch die Helligkeiten der Buchstaben ändert. Dies ist leider mit den bisher beschriebenen Mitteln noch nicht zu vermeiden. Hat man alle Farben ausgewählt, so gelangt man

mit dem Trigger ('Feuerknopf' am Joystick) zurück ins Hauptmenü. Das Ändern der Farben kann jederzeit vorgenommen werden, und muß nicht ganz am Anfang durchgeführt werden.

Laden

Hier kann ein Zeichensatz oder auch ein schon gezeichnetes Bild geladen werden. Auf die Frage 'Filename?' muß man einen Filenamen eingeben, wobei bei Diskettenzugriff das 'D:' nicht vergessen werden darf. Dann wird man erfragt, ob man das auf der Diskette befindliche Bild mitladen möchte. Antworten Sie mit 'J' oder 'N'.

ZSATZ

Wählt man diese Option an, so kommt man in Menü zwei. Hier sind alle 120 Zeichen, die das Programm nicht selbst verwendet, dargestellt. Außerdem steht ganz unten das Wort 'MENUE', um in das Hauptmenü zurückzukommen. Hier muß nun ein Buchstabe, der auf das Bild gebracht, oder geändert werden soll, ausgewählt werden. Diese Auswahl geschieht entweder dadurch, daß man den Trigger drückt, wenn der Cursor über dem gewählten Zeichen steht. Eine zweite Möglichkeit einen Buchstaben aufzunehmen, ist mit dem Cursor nach oben aus dem Textfenster zu gehen, und innerhalb des Bildes ein Zeichen aufzunehmen. Diese Möglichkeit wurde eingeräumt, da es nach dem Umdefinieren mehrerer Zeichen vorkommen kann, daß man das gewünschte Zeichen im Textfenster nicht wiedererkennt.

Wurde das Zeichen zum erstenmal angewählt, so muß man sich entscheiden, ob es normal oder 'invers' auf das Bild gebracht wird. D.h. man muß entweder die vierte oder die fünfte Farbe wählen. Dies geschieht, indem man durch Bewegen des Joysticks in irgendeine Richtung eines der beiden Farbfelder anwählt und dann den Trigger drückt.

Nun wird das Zeichen in der Matrix oben rechts dargestellt. Mit dem Zeichen befindet man sich nun irgendwo auf dem Bild, und das Zeichen kann auf dem Bild beliebig bewegt werden. Durch Drücken des Triggers kann das Zeichen an beliebiger Stelle und beliebig oft abgesetzt werden, so daß dadurch das Zeichnen eines ganzen Bildes sehr einfach wird.

Bewegt man den Cursor mit dem Zeichen in den Bereich der Matrix, so erscheint innerhalb der Matrix links unten ein eigener Cursor. Dieser Cursor kann innerhalb der Matrix

bewegt werden, und durch Drücken des Triggers wird das Zeichen verändert. Die Farbe des Cursor, kann durch hineinlaufen der an der linken Seite befindlichen Farbfelder geändert werden, wobei die Hintergrundfarbe, durch den kleinen senkrechten Strich gekennzeichnet ist. Läuft man mit dem Cursor auf den Pfeil nach unten, so gelangt man wieder ins Menü zwei zurück, und kann einen neuen Buchstaben auswählen.

In das Textfenster gelangt man auch, wenn man mit dem Cursor im Bild ganz nach unten geht.

ENDE

Damit kann das Programm beendet werden, d.h. der umdefinierte Zeichensatz, die aktuellen Farben und das Bild werden gespeichert. Weiterhin wird ein String gespeichert, der angibt, ob ein Zeichen mit der vierten oder mit der fünften Farbe gezeichnet wurde.

ORIGI

Damit kann ein Zeichen, das neu definiert wurde, in der alten Form wieder hergestellt werden. Diese Option dient dazu, ein Zeichen, das bereits umdefiniert wurde, aber für die spätere Anwendung im Textfenster gebraucht wird, wiederherzustellen. Diese Option kann nur angewendet werden, wenn sich ein Zeichen in der Matrix befindet. Nachdem das Zeichen in den Originalzustand zurückgeführt wurde, gelangt man in die Matrix, die man mit der Option 'Pfeil nach unten' sofort verlassen kann.

CLEAR

Diese Option unterscheidet sich von der obigen nur dadurch, daß das Zeichen gelöscht wird.

TOTAL

Die Option TOTAL gibt die Möglichkeit, in die Bereiche zu zeichnen, wo sich die Matrix und das Textfenster befinden. Da das Programm nur mit Joystick läuft, gestaltet sich das Zeichnen in dieser Option etwas mühsam. Man muß nämlich mit dem Joystick ein Zeichen aufnehmen, es über den Bildschirm bewegen, und kann es dann nur einmal an einer Stelle absetzen. Danach muß erneut ein Zeichen aufgenommen werden.

Da es nun passieren kann, daß das Zeichen im Cursor identisch mit dem Zeichen auf dem Bild ist, haben wir zur Unterscheidung in diesen Fällen zwei verschiedene Cursor, die angeben, ob das Zeichen im Cursor abgesetzt wird, oder ob das Hintergrundzeichen aufgenommen wird. Der Cursor blinkt dabei in der Farbe von Farbfeld vier falls das nächste Zeichen aufgenommen wird, und in der Farbe von Farbfeld fünf, falls das nächste Zeichen abgesetzt wird.

NEU

Mit dieser Option erreichen Sie einen Neustart des Programms. Damit es nicht passiert, daß mühevolle Arbeit aus Versehen zerstört wird, muß man diese Option über Tastatur bestätigen. Jede andere Eingabe außer 'J' führt in das Hauptmenü zurück.

Programmlisting

```
1000 REM ***********************************************
1010 REM *          ZEICHENSATZ NEU DEFINIEREN         *
1020 REM ***********************************************
1030 REM
1040 REM
1050 REM ***********************************************
1060 REM *       ZEILENNUMMERN DER UNTERPROGRAMME      *
1070 REM ***********************************************
1080 ASTOIN=6120
1090 BLINK=5800
1100 CHGET=6870
1110 COPYZ=5880
1120 INTOAS=5970
1130 IOERR=6310
1140 IOMENU=6400
1150 KNOPF=6980
1160 MENU1=6500
1170 MENU2=5240
1180 MOVEZ=5540
1190 MXCLR=5700
1200 OPINV=6730
1210 OPNORM=6590
1220 OUTMX=6040
1230 POSZA=6210
1240 ZITOMX=5360
1250 REM ***********************************************
1260 REM *                   VORSPANN                  *
1270 REM ***********************************************
```

```
1280 POKE 106,PEEK(740)-8
1290 GRAPHICS 12:REM s.ANHANG
1300 SETCOLOR 2,8,6
1310 DIM FNAME$(14)
1320 ZSATZ=(PEEK(106)+4)*256
1330 DIM NORI$(128)
1340 NORI$="0":NORI$(128)="0":NORI$(2)=NORI$
1350 DIM HILFE$(8)
1360 HILFE$="01041664"
1370 XALT=19:YALT=9
1380 DIM FARBEN$(5)
1390 FOR SCHLEIFE=1 TO 5
1400 FARBEN$(SCHLEIFE)=CHR$(PEEK(707+SCHLEIFE))
1410 NEXT SCHLEIFE
1420 OPEN #1,4,0,"K:"
1430 REM ***********************************************
1440 REM *   ZEICHENSATZ KOPIEREN UND VERAENDERN       *
1450 REM ***********************************************
1460 REM
1470 REM ----------- ZSATZ KOPIEREN --------------------
1480 DIM COPY$(33)
1490 FOR SCHLEIFE=1 TO 33
1500 READ BYTE:COPY$(SCHLEIFE)=CHR$(BYTE)
1510 NEXT SCHLEIFE
1520 DATA 104,104,133,209,104,133,208,133,206,169
1530 DATA 224,133,207,160,0,177,206,145,208,200
1540 DATA 208,249,230,209,230,207,165,207
1550 DATA 201,228,208,239,96
1560 VAR=USR(ADR(COPY$),ZSATZ)
1570 REM ------------- 8 NEUE ZEICHEN -------------------
1580 FOR SCHLEIFE=1 TO 8
1590 READ ZI:ZADR=ZSATZ+8*ZI
1600 FOR Y=0 TO 7
1610 READ BYTE:POKE ZADR+Y,BYTE
1620 NEXT Y
1630 NEXT SCHLEIFE
1640 DATA 64,0,0,0,0,0,0,0,0
1650 DATA 65,85,85,85,85,85,85,85,85
1660 DATA 66,170,170,170,170,170,170,170,170
1670 DATA 67,255,255,255,255,255,255,255,255
1680 DATA 68,85,85,0,0,0,0,0,0
1690 DATA 69,5,5,5,5,5,5,5,5
1700 DATA 70,0,3,3,51,51,15,3,0
1710 DATA 71,0,192,192,204,204,240,192,0
1720 POKE 756,ZSATZ/256
1730 REM ***********************************************
1740 REM *                MATRIX ZEICHNEN              *
1750 REM ***********************************************
1760 COLOR 5:PLOT 31,0:DRAWTO 31,7
1770 COLOR 4:PLOT 32,8:DRAWTO 39,8
1780 COLOR 5:PLOT 28,1
```

```
1790 COLOR 1:PLOT 29,2:PLOT 30,2
1800 COLOR 2:PLOT 29,3:PLOT 30,3
1810 COLOR 3:PLOT 29,4:PLOT 30,4
1820 COLOR 131:PLOT 29,5:PLOT 30,5
1830 COLOR 134:PLOT 29,7:COLOR 135:PLOT 30,7
1840 WAHL=1
1850 GOSUB MENU1
1860 GOTO 2070
1870 REM ***********************************************
1880 REM *            HAUPTPROGRAMM (MENUE1)           *
1890 REM ***********************************************
1900 GOSUB CHGET
1910 IF ST<>11 AND ST<>7 AND ST<>13 AND ST<>14 THEN 2080
1920 GOSUB OPNORM+10*WAHL
1930 IF ST<>11 THEN 1990
1940 WAHL=WAHL-1
1950 PRINT "←←←←←←←←←";
1960 IF WAHL=0 THEN WAHL=8:PRINT "↓↓";
1970 IF WAHL=4 THEN PRINT "↑↑";
1980 GOTO 2070
1990 IF ST<>7 THEN 2050
2000 WAHL=WAHL+1
2010 PRINT "→→→→→→→→→→";
2020 IF WAHL=9 THEN WAHL=1:PRINT "↑↑";
2030 IF WAHL=5 THEN PRINT "↓↓";
2040 GOTO 2070
2050 WAHL=WAHL+4:IF WAHL>8 THEN WAHL=WAHL-8
2060 ? "↑↑";
2070 GOSUB OPINV+10*WAHL
2080 GOSUB KNOPF
2090 IF TR=1 THEN 1900
2100 GOSUB OPNORM+10*WAHL
2110 IF WAHL>4 THEN 2130
2120 ON WAHL GOSUB 2250,2670,3250,4670:GOTO 2140
2130 ON WAHL-4 GOSUB 2950,3030,4880,3150
2140 GOSUB MENU1
2150 IF WAHL=1 THEN 2200
2160 FOR SCHLEIFE=2 TO WAHL
2170 ? "→→→→→→→→→→";
2180 NEXT SCHLEIFE
2190 IF WAHL>4 THEN PRINT "↓↓";
2200 GOSUB OPINV+10*WAHL
2210 GOTO 1900
2220 REM ***********************************************
2230 REM *          FARBEN AENDERN                     *
2240 REM ***********************************************
2250 PRINT "↖R/L       FARBE AENDERN"
2260 PRINT "O/U       NEUES FARBFELD"
2270 PRINT "TRIGGER  ZURUECK ZUR HAUPTAUSWAHL"
2280 FARBREG=712
2290 GOSUB MXCLR
```

```
2300 GOTO 2480
2310 GOSUB CHGET
2320 IF ST<>11 THEN 2360
2330 FARBE=FARBE-2
2340 IF FARBE<0 THEN FARBE=254
2350 GOTO 2390
2360 IF ST<>7 THEN 2410
2370 FARBE=FARBE+2
2380 IF FARBE>254 THEN FARBE=0
2390 WAIT=5
2400 GOTO 2500
2410 IF ST<>14 THEN 2450
2420 FARBREG=FARBREG-1
2430 IF FARBREG<708 THEN FARBREG=712
2440 GOTO 2480
2450 IF ST<>13 THEN 2500
2460 FARBREG=FARBREG+1
2470 IF FARBREG>712 THEN FARBREG=708
2480 FARBE=PEEK(FARBREG)
2490 GOTO 2520
2500 WAIT=WAIT-1
2510 IF WAIT>0 THEN 2560
2520 WAIT=5
2530 IF FARBE-16*INT(FARBE/16)>8 THEN 2550
2540 POKE FARBREG,FARBE+2:GOTO 2560
2550 POKE FARBREG,FARBE-2
2560 FOR SCHLEIFE=1 TO 10:NEXT SCHLEIFE
2570 POKE FARBREG,FARBE
2580 GOSUB KNOPF
2590 IF TR=1 THEN 2310
2600 FOR SCHLEIFE=1 TO 5
2610 FARBEN$(SCHLEIFE)=CHR$(PEEK(707+SCHLEIFE))
2620 NEXT SCHLEIFE
2630 RETURN
2640 REM ***********************************************
2650 REM *    LADEN EINES ZEICHENSATZES                *
2660 REM ***********************************************
2670 GOSUB IOMENU
2680 TRAP 2900
2690 OPEN #2,4,0,FNAME$
2700 INPUT #2;NORI$
2710 INPUT #2;FARBEN$
2720 FOR SCHLEIFE=1 TO 5
2730 POKE 707+SCHLEIFE,ASC(FARBEN$(SCHLEIFE))
2740 NEXT SCHLEIFE
2750 FOR SCHLEIFE=0 TO 1023
2760 GET #2,BYTE:POKE ZSATZ+SCHLEIFE,BYTE
2770 NEXT SCHLEIFE
2780 PRINT "5↓BILDSCHIRM NACHLADEN (J/N)"
2790 GET #1,TASTE
2800 IF TASTE<>ASC("J") THEN 2870
```

```
2810 PRINT "↖"
2820 VAR=PEEK(88)+256*PEEK(89)
2830 FOR SCHLEIFE=0 TO 960
2840 GET #2,BYTE
2850 POKE VAR+SCHLEIFE,BYTE
2860 NEXT SCHLEIFE
2870 CLOSE #2
2880 POP :GOTO 1760
2890 REM ---------------- I/O ERROR --------------------
2900 GOSUB IOERR
2910 GOTO 2670
2920 REM ***************************************************
2930 REM *                ORIGINALZEICHEN                  *
2940 REM ***************************************************
2950 IF ZA<8 THEN RETURN
2960 FOR SCHLEIFE=0 TO 7
2970 POKE ZSATZ+8*ZI+SCHLEIFE,PEEK(57344+8*ZI+SCHLEIFE)
2980 NEXT SCHLEIFE
2990 GOTO 3080
3000 REM ***************************************************
3010 REM *             ZEICHEN LOESCHEN                    *
3020 REM ***************************************************
3030 IF ZA<8 THEN RETURN
3040 ZADR=ZSATZ+8*ZI
3050 FOR SCHLEIFE=0 TO 7
3060 POKE ZADR+SCHLEIFE,0
3070 NEXT SCHLEIFE
3080 GOSUB ZITOMX
3090 GOSUB MENU2
3100 GOSUB POSZA
3110 GOTO 3970
3120 REM ***************************************************
3130 REM *           NEU BEGINNEN                          *
3140 REM ***************************************************
3150 POKE 764,255
3160 PRINT "↖↓BITTE 'J' TIPPEN UM NEU ZU BEGINNEN"
3170 GET #1,TASTE
3180 IF TASTE=ASC("J") THEN RUN
3190 RETURN
3200 REM ***************************************************
3210 REM *       ZEICHNEN UND ZSATZ AENDERN                *
3220 REM ***************************************************
3230 REM
3240 REM ------------- BUCHSTABEN ZEICHNEN --------------
3250 GOSUB MENU2
3260 POKE 656,0:POKE 657,39
3270 GOTO 3300
3280 REM ------------ BUCHSTABEN AUSWAEHLEN -------------
3290 IF STICK(0)<>15 THEN 3290
3300 POKE 752,0:PRINT "→";
3310 POKE 77,0
```

```
3320 GOSUB CHGET
3330 IF ST=11 THEN PRINT "←";:GOTO 3520
3340 IF ST=7 THEN PRINT "→";:GOTO 3520
3350 IF ST<>13 THEN 3460
3360 IF PEEK(656)<2 THEN PRINT "↓";:GOTO 3520
3370 IF PEEK(656)=3 THEN 3420
3380 SPALTE=PEEK(657)
3390 POKE 752,1:PRINT
3400 GOSUB OPINV
3410 GOTO 3520
3420 PRINT "↑":GOSUB OPNORM
3430 POKE 657,SPALTE
3440 POKE 752,0:PRINT "↓";
3450 GOTO 3520
3460 IF ST<>14 THEN 3520
3470 IF PEEK(656)=0 THEN 4440
3480 IF PEEK(656)<3 THEN PRINT "↑";:GOTO 3520
3490 PRINT "↑":GOSUB OPNORM
3500 POKE 657,SPALTE
3510 POKE 752,0:PRINT "↑";
3520 FOR SCHLEIFE=1 TO 2:NEXT SCHLEIFE
3530 GOSUB KNOPF
3540 IF TR=1 THEN 3320
3550 IF PEEK(656)=3 THEN RETURN
3560 ZI=PEEK(665)
3570 POKE 752,1:PRINT "←";
3580 GOSUB INTOAS
3590 REM -------------- NORMAL ODER INVERS --------------
3600 IF NORI$(ZI+1,ZI+1)<>"0" THEN 3820
3610 GOSUB KNOPF:IF TR=0 THEN 3610
3620 WAIT=1
3630 GOTO 3670
3640 GOSUB CHGET
3650 IF ST=15 THEN 3680
3660 IF COL5=0 THEN COL4=0:COL5=131:GOTO 3680
3670 COL4=3:COL5=0
3680 WAIT=WAIT-1
3690 IF WAIT>0 THEN 3740
3700 WAIT=4
3710 COLOR 0
3720 PLOT 29,4:PLOT 30,4
3730 PLOT 29,5:PLOT 30,5
3740 FOR SCHLEIFE=1 TO 8:NEXT SCHLEIFE
3750 COLOR COL4:PLOT 29,4:PLOT 30,4
3760 COLOR COL5:PLOT 29,5:PLOT 30,5
3770 GOSUB KNOPF
3780 IF TR=1 THEN 3640
3790 IF COL4=0 THEN NORI$(ZI+1,ZI+1)="I":GOTO 3820
3800 NORI$(ZI+1,ZI+1)="N"
3810 REM ------------- ZEICHEN DARSTELLEN --------------
3820 GOSUB ZITOMX
```

```
3830 IF NORI$(ZI+1,ZI+1)="I" THEN ZA=ZA+128
3840 X=XALT:Y=YALT
3850 FARBE=131
3860 GOTO 3910
3870 COLOR ALTZA:PLOT X,Y
3880 GOSUB MOVEZ
3890 IF Y>19 THEN 3290
3900 IF X>28 AND Y<9 THEN 3960
3910 GOSUB BLINK
3920 GOSUB KNOPF
3930 IF TR=0 THEN ALTZA=ZA:XALT=X:YALT=Y
3940 GOTO 3870
3950 REM ------- ZEICHEN VERAENDERN -------
3960 ZADR=ZSATZ+8*ZI
3970 CURSOR=0
3980 XMX=32:YMX=7
3990 GOTO 4230
4000 GOSUB CHGET
4010 IF ST=15 THEN 4260
4020 COLOR PIXEL:PLOT XMX,YMX:PLOT XMX+1,YMX
4030 IF ST<>11 THEN 4120
4040 IF XMX>32 THEN XMX=XMX-2:GOTO 4230
4050 IF YMX=0 OR YMX=6 THEN 4260
4060 IF YMX=4 AND NORI$(ZI+1,ZI+1)="I" THEN 4260
4070 IF YMX=7 THEN 3290
4080 IF YMX<>5 THEN CURSOR=YMX-1:GOTO 4230
4090 IF NORI$(ZI+1,ZI+1)="N" THEN 4260
4100 CURSOR=131
4110 GOTO 4230
4120 IF ST<>7 THEN 4160
4130 IF XMX=38 THEN 4260
4140 XMX=XMX+2
4150 GOTO 4230
4160 IF ST<>13 THEN 4200
4170 IF YMX=7 THEN YMX=0:GOTO 4230
4180 YMX=YMX+1
4190 GOTO 4230
4200 IF ST<>14 THEN 4260
4210 IF YMX=0 THEN YMX=7:GOTO 4230
4220 YMX=YMX-1
4230 POSITION XMX,YMX:GET #6,PIXEL
4240 COLOR CURSOR:PLOT XMX,YMX:PLOT XMX+1,YMX
4250 WAIT=1
4260 WAIT=WAIT-1
4270 IF WAIT>0 THEN 4320
4280 WAIT=4
4290 IF PIXEL=CURSOR THEN COLOR 4:GOTO 4310
4300 COLOR PIXEL
4310 PLOT XMX,YMX:PLOT XMX+1,YMX
4320 FOR SCHLEIFE=1 TO 2:NEXT SCHLEIFE
4330 COLOR CURSOR:PLOT XMX,YMX:PLOT XMX+1,YMX
```

```
4340 GOSUB KNOPF
4350 IF TR=1 THEN 4000
4360 BYTE=VAL(HILFE$(39-XMX,40-XMX))
4370 VAR=CURSOR-PIXEL
4380 IF VAR>3 THEN VAR=VAR-128:GOTO 4400
4390 IF VAR<-3 THEN VAR=VAR+128
4400 POKE ZADR+YMX,PEEK(ZADR+YMX)+VAR*BYTE
4410 PIXEL=CURSOR
4420 GOTO 4000
4430 REM ------------------- AUFNEHMEN ------------------
4440 ZI=0
4450 GOSUB INTOAS
4460 GOSUB MXCLR
4470 X=PEEK(657):Y=19
4480 POKE 752,1:PRINT "←";
4490 FARBE=3
4500 GOTO 4540
4510 COLOR ALTZA:PLOT X,Y
4520 GOSUB MOVEZ
4530 IF Y>19 OR X>28 AND Y<9 THEN 3290
4540 GOSUB BLINK
4550 GOSUB KNOPF
4560 IF TR=1 THEN 4510
4570 COLOR ALTZA:PLOT X,Y
4580 XALT=X:YALT=Y
4590 ZA=ALTZA
4600 GOSUB ASTOIN
4610 GOSUB ZITOMX
4620 GOSUB POSZA
4630 GOTO 3850
4640 REM ***********************************************
4650 REM *                ABSPEICHERN                  *
4660 REM ***********************************************
4670 GOSUB IOMENU
4680 TRAP 4830
4690 OPEN #2,8,0,FNAME$
4700 GOSUB OUTMX
4710 PRINT #2;NORI$
4720 PRINT #2;FARBEN$
4730 FOR SCHLEIFE=0 TO 1023
4740 PUT #2,PEEK(ZSATZ+SCHLEIFE)
4750 NEXT SCHLEIFE
4760 VAR=PEEK(88)+256*PEEK(89)
4770 FOR SCHLEIFE=0 TO 960
4780 PUT #2,PEEK(VAR+SCHLEIFE)
4790 NEXT SCHLEIFE
4800 CLOSE #2
4810 GRAPHICS 0:END
4820 REM ------------------- I/O ERROR ------------------
4830 GOSUB IOERR
4840 GOTO 4670
```

```
4850 REM ***************************************************
4860 REM *          NUR NOCH ZEICHNEN                      *
4870 REM ***************************************************
4880 GOSUB IOMENU
4890 TRAP 5140
4900 OPEN #2,8,0,FNAME$
4910 GOSUB OUTMX
4920 GRAPHICS 60:POKE 756,ZSATZ/256
4930 FOR SCHLEIFE=1 TO 5
4940 POKE 707+SCHLEIFE,ASC(FARBEN$(SCHLEIFE))
4950 NEXT SCHLEIFE
4960 X=XALT:Y=YALT
4970 POSITION X,Y:GET #6,ALTZA
4980 ZA=32:FARBE=3
4990 GOSUB COPYZ
5000 GOSUB KNOPF
5010 IF TR=1 THEN 4990
5020 ZA=ALTZA
5030 GOSUB KNOPF
5040 IF TR=0 THEN 5030
5050 FARBE=131
5060 GOSUB COPYZ
5070 GOSUB KNOPF
5080 IF TR=1 THEN 5060
5090 ALTZA=ZA
5100 GOSUB KNOPF
5110 IF TR=0 THEN 5100
5120 GOTO 4980
5130 REM ------------------- I/O ERROR -----------------
5140 GOSUB IOERR
5150 GOTO 4880
5160 REM ***************************************************
5170 REM *           KURZE UNTERPROGRAMME                  *
5180 REM ***************************************************
5190 REM
5200 REM
5210 REM ***************************************************
5220 REM * -MENU2-            ZEICHENMENUE                 *
5230 REM ***************************************************
5240 POKE 82,0:POKE 83,39
5250 PRINT "{";
5260 POKE 766,1
5270 FOR SCHLEIFE=8 TO 127
5280 PRINT CHR$(SCHLEIFE);
5290 NEXT SCHLEIFE
5300 GOSUB OPNORM
5310 POKE 766,0
5320 RETURN
5330 REM ***************************************************
5340 REM * -ZITOMX-      ZEICHNE ZI IN DIE MATRIX          *
5350 REM ***************************************************
```

```
5360 IF NORI$(ZI+1,ZI+1)="N" THEN COL4=3:COL5=0:GOTO 5390
5370 COL4=0:COL5=131
5380 COLOR COL4:PLOT 29,4:PLOT 30,4
5390 COLOR COL5:PLOT 29,5:PLOT 30,5
5400 FOR YMX=0 TO 7
5410 BYTE=PEEK(ZSATZ+8*ZI+YMX)
5420 FOR XMX=0 TO 3
5430 PIXEL=BYTE-4*INT(BYTE/4)
5440 BYTE=INT(BYTE/4)
5450 IF PIXEL=3 AND NORI$(ZI+1,ZI+1)="I" THEN PIXEL=131
5460 COLOR PIXEL
5470 PLOT 38-2*XMX,YMX:PLOT 39-2*XMX,YMX
5480 NEXT XMX
5490 NEXT YMX
5500 RETURN
5510 REM ***********************************************
5520 REM * -MOVEZ-       ZEICHEN BEWEGEN              *
5530 REM ***********************************************
5540 GOSUB CHGET
5550 GOTO 5510+10*ST
5560 GOSUB 5580:GOTO 5640
5570 GOSUB 5580:GOTO 5650
5580 X=X+1:IF X>39 THEN X=0
5590 RETURN
5600 GOSUB 5620:GOTO 5640
5610 GOSUB 5620:GOTO 5650
5620 X=X-1:IF X<0 THEN X=39
5630 RETURN
5640 Y=Y+1:RETURN
5650 Y=Y-1:IF Y<0 THEN Y=0
5660 RETURN
5670 REM ***********************************************
5680 REM * -MXCLR-     MATRIX LOESCHEN                 *
5690 REM ***********************************************
5700 COLOR 3:PLOT 29,4:PLOT 30,4
5710 COLOR 131:PLOT 29,5:PLOT 30,5
5720 COLOR 0
5730 FOR Y=0 TO 7
5740 PLOT 32,Y:DRAWTO 39,Y
5750 NEXT Y
5760 RETURN
5770 REM ***********************************************
5780 REM * -BLINK-            BLINKEN                  *
5790 REM ***********************************************
5800 POSITION X,Y:GET #6,ALTZA
5810 IF ZA=ALTZA THEN COLOR FARBE:GOTO 5830
5820 COLOR ZA
5830 PLOT X,Y
5840 RETURN
5850 REM ***********************************************
5860 REM * -COPYZ-       ZEICHEN KOPIEREN             *
```

```
5870 REM ***************************************************
5880 COLOR ALTZA:PLOT X,Y
5890 IF PEEK(53279)=6 THEN POP :GOTO 4710
5900 GOSUB MOVEZ
5910 IF Y>23 THEN Y=23
5920 GOSUB BLINK
5930 RETURN
5940 REM ***************************************************
5950 REM * -INTOAS-  INTERNEN CODE IN ATASCII UMWANDELN *
5960 REM ***************************************************
5970 IF ZI<64 THEN ZA=ZI+32:GOTO 6000
5980 IF ZI<96 THEN ZA=ZI-64:GOTO 6000
5990 ZA=ZI
6000 RETURN
6010 REM ***************************************************
6020 REM * -OUTMX-        MATRIX AUSBLENDEN               *
6030 REM ***************************************************
6040 COLOR 0:PLOT 28,1
6050 FOR Y=0 TO 8
6060 PLOT 29,Y:DRAWTO 39,Y
6070 NEXT Y
6080 RETURN
6090 REM ***************************************************
6100 REM * -ASTOIN-  ATASCII IN INTERNEN CODE UMWANDELN *
6110 REM ***************************************************
6120 VAR=ZA
6130 IF ZA>127 THEN VAR=VAR-128
6140 IF VAR<32 THEN ZI=VAR+64:GOTO 6170
6150 IF VAR<96 THEN ZI=VAR-32:GOTO 6170
6160 ZI=VAR
6170 RETURN
6180 REM ***************************************************
6190 REM * -POSZA-    POSITION VON ZA BESTIMMEN           *
6200 REM ***************************************************
6210 VAR=ZA-8
6220 IF VAR>127 THEN VAR=VAR-128
6230 POKE 656,INT(VAR/40)
6240 VAR=VAR-40*INT(VAR/40)
6250 IF VAR=0 THEN POKE 657,39:GOTO 6270
6260 POKE 657,VAR-1
6270 RETURN
6280 REM ***************************************************
6290 REM * -IOERR-         FEHLER BEIM I/O                 *
6300 REM ***************************************************
6310 CLOSE #1
6320 IF PEEK(195)=130 THEN POP :RETURN
6330 PRINT "↰↓FEHLER ";PEEK(195)
6340 PRINT "↓TASTE DRUECKEN"
6350 GET #1,TASTE
6360 RETURN
6370 REM ***************************************************
```

```
6380 REM * -IOMENU-        MENUE FUER I/O                    *
6390 REM ***********************************************************
6400 POKE 764,255
6410 POKE 752,0
6420 PRINT "↖↓FILENAME"
6430 PRINT "↓RETURN =ZURUECK←←←←←↑↑";
6440 INPUT FNAME$
6450 POKE 752,1
6460 RETURN
6470 REM ***********************************************************
6480 REM * -MENU1-          ZEILEN FUER MENUES                    *
6490 REM ***********************************************************
6500 POKE 82,4:POKE 83,39
6510 POKE 752,1
6520 PRINT "↖FARBE    LADEN    ZSATZ    ENDE"
6530 PRINT :PRINT "ORIGI    CLEAR    TOTAL    NEU"
6540 PRINT "↓";
6550 RETURN
6560 REM ***********************************************************
6570 REM * -OPNORM-      OPTION NORMALSCHRIFT                    *
6580 REM ***********************************************************
6590 PRINT "                    MENUE";:RETURN
6600 PRINT "FARBE";:GOTO 6680
6610 PRINT "LADEN";:GOTO 6680
6620 PRINT "ZSATZ";:GOTO 6680
6630 PRINT "ENDE ";:GOTO 6680
6640 PRINT "ORIGI";:GOTO 6680
6650 PRINT "CLEAR";:GOTO 6680
6660 PRINT "TOTAL";:GOTO 6680
6670 PRINT "NEU  ";
6680 PRINT "←←←←←";
6690 RETURN
6700 REM ***********************************************************
6710 REM * -OPINV-            OPTION INVERSE SCHRIFT              *
6720 REM ***********************************************************
6725 REM DIE FOLGENDEN WOERTER IN DIESEM UNTERPROGRAMM
6726 REM  I N V E R S  SCHREIBEN
6730 PRINT "                    MENUE";:RETURN
6740 PRINT "FARBE";:GOTO 6820
6750 PRINT "LADEN";:GOTO 6820
6760 PRINT "ZSATZ";:GOTO 6820
6770 PRINT "ENDE ";:GOTO 6820
6780 PRINT "ORIGI";:GOTO 6820
6790 PRINT "CLEAR";:GOTO 6820
6800 PRINT "TOTAL";:GOTO 6820
6810 PRINT "NEU  ";
6820 PRINT "←←←←←";
6830 RETURN
6840 REM ***********************************************************
6850 REM * -CHGET-       TASTATURABFRAGE                          *
6860 REM ***********************************************************
```

```
6870 IF PEEK(764)=255 THEN 6930
6880 GET #1,TASTE
6890 IF TASTE=43 THEN ST=11:RETURN
6900 IF TASTE=42 THEN ST=7:RETURN
6910 IF TASTE=45 THEN ST=14:RETURN
6920 IF TASTE=61 THEN ST=13:RETURN
6930 ST=15
6940 RETURN
6950 REM ***********************************************
6960 REM * -KNOPF-      SELECT ABFRAGEN                *
6970 REM ***********************************************
6980 TR=(PEEK(53279)<>5)
6990 RETURN
```

Programmbeschreibung

Zunächst die Programmaufteilung nach Zeilennummern:

1000 Zeilennummern der Unterprogramme
1250 Vorspann
1430 Zeichensatz kopieren und ändern
1730 Matrix zeichnen
1870 Menü 1 (Hauptprogramm)
2220 Farben ändern
2640 Laden eines Zeichensatzes
2920 Originalzeichen
3000 Zeichen löschen
3120 Neustart
3200 Zeichnen und Zeichensatz ändern
    3240 Buchstaben zeichnen
    3280 Buchstaben auswählen
    3590 Normal oder Invers
    3810 Zeichen darstellen
    3950 Zeichen verändern
    4430 Zeichen aufnehmen
4640 Zeichensatz, Farben und Bild speichern
4850 Zeichnen in Hilfsmatrix und Textfenster

Unterprogramme

5210 Zeichenmenü
5330 Zeichne ZI in Matrix
5510 Zeichen bewegen
5670 Matrix löschen
5770 Blinken
5850 Zeichen kopieren
5940 Internen Code in ATASCII umwandeln
6010 Matrix ausblenden

6090 ATASCII-Code in internen Code umwandeln
6180 Position von ZA bestimmen
6280 Fehler in der Ein-/Ausgabe
6370 Menü für Ein-/Ausgabe
6470 Zeilen für Menüs
6560 Optionen Normalschrift
6700 Optionen inverse Schrift
6840 Tastaturabfrage
6950 SELECT-Taste abfragen

Zeilennummern der Unterprogramme

Da in unserem Programm viele kleine Unterprogramme aufgerufen werden, haben wir diese mit Namen benannt. Die Adressen werden den entsprechenden Variablen zugeordnet. Dies hat den Vorteil, daß das Programm leichter lesbar wird.

Vorspann

In der Zeile 1280 schaffen wir uns zunächst Platz für unseren Zeichensatz. Dabei ziehen wir nicht vom Wert in RAMTOP die 8*256 Byte ab, sondern von dem Inhalt in RAMSIZ (Adresse 740). Der Wert in RAMTOP wird sonst nämlich bei jedem Programmstart erneut um 8 erniedrigt, wenn nicht bei jedem Programmabruch RESET gedrückt wird.

Sodann folgt die Dimensionierung einiger Strings, die später erläutert werden. In Zeile 1340 wird der String NORI$ initialisiert, d.h. der gesamte String wird mit 'O' vollgeschrieben. Wir kommen auf diese Methode in dem Kapitel über Basic noch einmal zurück. Der String FARBEN$ wird mit den aktuellen Farbwerten belegt, die immer mit gespeichert und geladen werden.

XALT und YALT sind Koordinaten, die zu Beginn mit der Position in der Mitte des Bildschirms belegt werden.

In Zeile 1420 wird schließlich noch ein File zur Tastatur geöffnet, da bei einigen Optionen eine Bestätigung erforderlich ist.

Zeichensatz kopieren und verändern

Damit das Kopieren des Zeichensatzes vom ROM ins RAM nicht zulange dauert, machen wir dies mit einem Maschinenprogramm. Dieses Maschinenprogramm schreiben wir in den String COPY$. Nach dem Kopieren, definieren wir uns acht

Zeichen. Die Datazeilen in den Zeilen 1640 bis 1710 haben als ersten Wert die Nummer des Zeichens (interner Code), sodann folgen acht Byte für das Bitmuster. Diese Zeichen dienen uns als Cusor, bzw. als Begrenzungslinien für die Matrix und zwei Zeichen für den Pfeil. In Zeile 1720 schalten wir schließlich auf den neuen Zeichensatz um.

Bild zeichnen

Obwohl der Grafikmode 12 ein Textmode ist, arbeiten wir hier mit PLOT und DRAWTO. Dies hat den Vorteil, daß im COLOR-Befehl die Nummer des Zeichens angegeben ist (ATASCII).

Dann wird das Menü gezeichnet. Dies geschieht im Unterprogramm MENU1. Mit dem Befehl POKE 752,1 schalten wir den Cursor aus. Die Register LMARGN (Adresse 82) und RMARGN (Adresse 83) beinhalten die Werte für den linken, bzw. rechten Rand, wobei der Wert null ganz links ist. Wir schaffen mit diesen Befehlen ein Textfenster, das 36=4*9 Zeichen breit ist. Dies benötigen wir, um bequem vom rechten auf den linken Rand zu kommen. Da der POSITION-Befehl im Textfenster nicht arbeitet, müssen wir alle Cursorpositionierungen über die Cursor-Befehle machen.

Hauptprogramm

Das Hauptprogramm dient zur Auswahl einer Option, und springt diese dann mit GOSUB an. Die Variable WAHL ist dabei die grundlegende Größe, denn diese gibt die Nummer der Option an. Wurde der Joystick in eine der vier erlaubten Richtungen bewegt (Zeilen 1900 und 1910), so wird in Zeile 1920 die aktuelle Option zunächst wieder in Normalschrift geschrieben und die Variable WAHL dann in den Zeilen 1930 bis 2050 auf den neuen Wert geändert. In Zeile 2070 wird die neue Option invers geschrieben. Wurde der Knopf gedrückt, so wird in den Zeilen 2110 bis 2130 das entsprechende Unterprogramm aufgerufen.

Nach Verlassen des Unterprogramms wird in den Zeilen 2140 bis 2200 das Menü gezeichnet und der Cursor in Abhängigkeit der Variablen WAHL auf die entprechende Option positioniert. Da jede FOR...NEXT-Schleife in Basic mindestens einmal durchlaufen wird, müssen wir eine spezielle Abfrage für den Fall WAHL=1 durchführen (Zeile 2150).

Farben ändern

Zunächst beschreiben wir das Textfenster mit den Anweisungen. Danach beginnt das Ändern der Werte in den Farbregistern 708 bis 712 (Variable FARBREG). FARBREG wird zuerst auf 712 gesetzt, d.h. auf das Register für die Hintergrundfarbe und die Matrix wird gelöscht. In den Zeilen 2310 bis 2490 wird je nach Joystick-Bewegung entweder ein Wert in einem Farbregister geändert, oder ein anderes Farbregister angewählt.

Das Blinken der Farben wird in den Zeilen 2530 bis 2570 erzeugt. Damit die Farben nicht zu schnell blinken, wird die Variable WAIT heruntergezählt. Das Blinken selbst wird erreicht, indem die Helligkeit um eine Stufe geändert wird, d.h. der Wert im Register wird um zwei Einheiten geändert. Dabei soll die Farbe beim Blinken heller werden, wenn die Helligkeit kleiner oder gleich acht ist und umgekehrt. Die Zerlegung der Farbe in Farbwert und Helligkeit und die Abfrage erfolgt in Zeile 2530.

Bevor die Option 'FARBE' verlassen wird übernehmen wir in den Zeilen 2600 bis 2620 die aktuellen Farbwerte in die Zeichenreihe FARBEN$.

Laden eines Zeichensatzes

Als erstes springen wir in das Unterprogramm IOMENU das als Ausgabeparameter einen Filenamen übergibt. Nun eröffnen wir ein File zum Lesen mit diesem Filenamen. In den Zeilen 2750 bis 2770 lesen wir mit dem GET-Befehl die 1024 Byte für den Zeichensatz ein und schreiben diese Zeichen an die entsprechende Stelle ins RAM. Wahlweise wird dann der Bildschirm dazu geladen. Da ein Bildschirm im Grafikmode 12 immer 40x24=960 Zeichen groß ist (egal, ob mit Textfenster oder ohne) lesen wir diese 960 Byte mit dem GET-Befehl. Die Anfangsadresse des Bildschirm-RAM's steht im Register SAVMSC (Adresse 88,89). Mit dem Befehl CLOSE#1 schließen wir die Datei wieder. Da wir durch das Laden des Bildschirms unsere Matrix überschrieben haben, können wir nicht mit RETURN in das Hauptprogramm zurück springen. Weil sich der Computer bei einem GOSUB-Befehl die Rücksprungadresse merkt, um bei einem RETURN-Befehl zurückzufinden, sollte man Unterprogramme nicht mit GOTO verlassen, ohne vorher die Rücksprungadresse zu löschen. Dies geschieht mit dem POP-Befehl. Wir springen dann an die

Stelle, wo die Matrix gezeichnet wird.

Originalzeichen

In Zeile 2950 fragen wir zunächst ab, ob schon ein Zeichen angewählt wurde. Damit vermeiden wir, daß eines der oben definierten Zeichen (ATASCII-Wert 0-7) verändert wird. In den Zeilen 2960 bis 2980 kopieren wir das Original-Bitmuster an die entsprechende Stelle in unseren Zeichensatz. Danach verzweigen wir zu Zeile 3080.

Zeichen löschen

In Zeile 3030 fragen wir ab, ob schon ein Zeichen angewählt ist. In den Zeilen 3040 bis 3070 wird dann das Bitmuster dieses Zeichens in unserem Zeichensatz mit Nullen vollgeschrieben. In den Zeilen 3080 bis 3100 rufen wir Unterprogramme auf, die das Zeichen (Leerzeichen bzw. Originalzeichen aus obigem Unterprogramm) in die Matrix bringen, das Menü 2 aufrufen, und den Cursor an die richtige Stelle in diesem Menü plazieren. Dann verzweigen wir zu Zeile 3970, wo die neue Definition des Zeichens durchgeführt wird.

Neu beginnen

Diese Option gestattet es, das Programm neu zu starten. Damit nicht versehentlich alles gelöscht wird, muß man den Neustart über die Tastatur bestätigen. Mit dem GET-Befehl warten wir, bis eine Taste gedrückt ist. Die Abfrage, ob 'J' gedrückt wurde geschieht in Zeile 3180. Wurde eine andere Taste gedrückt, so gehen wir mit einem RETURN-Befehl ins Hauptprogramm zurück. Dabei kann ein Problem auftreten: wurden während des Programmlaufs Tasten betätigt, so merkt sich der Computer die jeweils letzte Taste, bis diese über einen entsprechenden Befehl abgefragt wird. So ein Befehl ist auch der GET-Befehl. Um nun zu verhindern, daß eine unerwünschte Taste mit dem GET-Befehl gelesen wird, müssen wir den Tastaturpuffer löschen. Dazu muß man in die Variable CH (Adresse 764) den Wert 255 schreiben (Zeile 3150).

Zeichnen und Zeichensatz ändern

Zunächst einmal zwei Vorbemerkungen: wie schon erwähnt arbeitet im Textfenster der POSITION-Befehl nicht. Man kann diesen Befehl umgehen, indem man die Koordinaten des

Cursors direkt in die entsprechenden Register schreibt. Diese Register sind 656 (Zeile) und 657 (Spalte). Wir belegen diese mit '0' für die Reihe und '39' für die Spalte vor (rechts oben im Textfenster).

Mit den Befehlen POKE 752,0 bzw. POKE 752,1 schaltet man den Cursor im Textfenster an bzw. aus. Dies geschieht jedoch nicht sofort, sondern erst bei der nächsten Ausgabe auf den Bildschirm. Da wir den Cursor an- und ausschalten müssen, geben wir nach dem Ausschalten immer einen Cursor-nach-links-Befehl, und nach dem Einschalten einen Cursor-nach-rechts-Befehl, damit ist die ursprüngliche Position wieder erreicht. Diese Bewegungen sind erforderlich, um das Ein-/Ausschalten tatsächlich durchzuführen.

In Zeile 3250 rufen wir das Unterprogramm MENU2 auf, das alle Buchstaben, bis auf die acht anfangs definierten, auf den Bildschirm zeichnet und die Option 'MENUE' auf den Bildschirm schreibt. In Zeile 6115 schalten wir den Cursor ein und gehen ein Feld nach rechts. Damit steht der Cursor sichtbar an der Position 0,0, da wir die Spalte mit dem Wert 39 vorbelegt haben. Der Befehl POKE 77,0 dient dazu, den ATTRACT-Mode (das ist das zufällige Ändern der Farben, wenn ca. neun Minuten keine Taste bedient wurde) auszuschalten.

Dieser Befehl sollte immer dann verwendet werden, wenn ein Programm nicht mit der Tastatur arbeitet. Jedoch muß man darauf achten, daß der ATTRACT-Mode eingeschaltet wird, wenn der Benutzer nicht mehr mit dem Programm arbeitet. Wir verwenden den Befehl an der Stelle, wo ein neuer Buchstabe ausgewählt wird.

In den Zeilen 3320 bis 3400 wird der Cursor mittels Joystick bewegt, wobei beim Senkrechtgehen das Wort 'Menue' immer wieder invers geschrieben werden muß. Da hierbei die horizontale Position des Cursors verändert wird, müssen wir diese vorher speichern. Dies geschieht in Zeile 3380. In Zeile 3390 schalten wir den Cursor aus und geben einen PRINT-Befehl ohne Strichpunkt, damit der Cursor an den Anfang der vierten Zeile geht. Dann invertieren wir das Wort 'MENUE'. Der umgekehrte Vorgang funktioniert analog in den Zeilen 3420 bis 3440. Wird der Cursor oben aus dem Textfenster herausbewegt, so verzweigen wir in Zeile 3470 zu Zeile 4440.

Wurde der Knopf gedrückt, so springen wir zurück ins Hauptprogramm, falls der Cursor sich in der vierten Zeile befand (Zeile 3550), andernfalls übernehmen wir den Buchstaben unter dem Cursor in die Variable ZI, schalten den Cursor aus, und gehen einen Schritt nach links. Die Adres-

se 665 enthält immer den Wert unter dem Cursor, d.h. das Zeichen das an der Cursorposition steht (in nicht inverser Darstellung). Der dabei erhaltene Wert, ist der interne Code dieses Zeichens. Diesen internen Code wandeln wir im Unterprogramm INTOAS in den entsprechenden ATASCII-Wert um.

Da ein Zeichen nur vier von den fünf möglichen Farben haben kann - je nach Darstellung 'normal' oder 'invers' -, müssen wir uns für eine Möglichkeit entscheiden (Zeilen 3590 bis 3800). Diese Entscheidung wird dann in den String NORI$ übernommen, in dem zunächst für jedes noch nicht definierte Zeichen eine 'O' steht. In Zeile 3600 fragen wir ab, ob das Zeichen schon als normal oder invers definiert ist. Ist dies der Fall, so verlassen wir diesen Programmteil und gehen zu Zeile 3820. Zeile 3610 ist eine Schleife, die erst dann verlassen wird, wenn der Knopf nicht mehr gedrückt ist. Dies ist notwendig, damit nicht aus versehen eine Farbe gewählt wird, weil der Knopf noch gedrückt ist. In den Zeilen 3660 und 3670 vertauschen wir jeweils die Farbwerte, falls der Joystick in irgendeine Richtung bewegt wird. In den Zeilen 3710 bis 3730 werden beide Farbfelder mit der Hintergrundfarbe überschrieben, so daß bei der gerade gesetzten Farbe ein Blinkeffekt entsteht. Nach einer kleinen Schleife werden die ursprünglichen Farben wieder hergestellt. Wurde der Knopf gedrückt, so wird in den Zeilen 3790 und 3800 ein 'I' in den NORI-String geschrieben, falls das Zeichen invers dargestellt werden muß, und ein 'N' für normale Darstellung.

Das ausgewählte Zeichen wird nun mit Hilfe des Unterprogramms ZITOMX in die Matrix gebracht. Muß das Zeichen invers dargestellt werden, so wird der ATASCII-Wert des Zeichen (ZA) um 128 erhöht, das bedeutet einen Übergang vom normalen Zeichen in das entsprechende inverse. In den Zeilen 3870 bis 3940 bewegen wir das Zeichen über den Bildschirm, wobei es abgesetzt werden kann. Diese Schleife wird verlassen, falls man mit dem Zeichen ins Textfenster geht (Zeile 3890) oder in die Matrix (Zeile 3900).

Der Programmteil von Zeile 3950 bis 4420 umfaßt das Verändern des aktuellen Zeichens innerhalb der Matrix.

Zuerst wird in Zeile 3960 die Adresse des Zeichens im Zeichensatz bestimmt. In den Zeilen 4000 bis 4220 werden die Koordinaten des Cursors innerhalb der Matrix je nach Joystickbewegung geändert. Dabei werden in den Zeilen 4050 bis 4110 die möglichen Optionen behandelt. In Zeile 4230 wird bestimmt welche Farbe sich an der neuen Cursorkoordinate befindet. Diese Abfrage geschieht mit dem GET-Befehl, wobei keine Datei geöffnet werden muß, weil Basic

die Datei #6 immer für den Bildschirm verwendet (außer im Grafikmode 0). In Zeile 4240 kann dann der Cursor gesetzt werden.

In den Zeilen 4290 bis 4330 vertauschen wir kurzzeitig die Cursorfarbe mit der Hintergrundfarbe, um so ein Blinken zu erzeugen. Sind Cursorfarbe und Hintergrundfarbe gleich, so würde man nichts erkennen. Deshalb lassen wir in diesem Fall unseren waagerechten Begrenzungsstrich kurz aufblinken. Selbstverständlich kann man sich hier ein eigenes Zeichen, z.B. ein Kreuz, definieren, doch wollten wir nicht zu viele Zeichen umdefinieren.

Wurde der Knopf gedrückt, so muß man im Zeichensatz zwei Bit ändern. Die ursprünglichen zwei Bit entsprechen dem Wert in der Variablen PIXEL, wobei man für den Wert 128 das Bitmuster 11 bekommt. Das neue Bitmuster steht in der Variablen CURSOR, wieder mit der 11 statt 128. Also muß man das ursprüngliche Bitmuster um den Wert CURSOR minus PIXEL ändern. Die Differenz aus den beiden Werten erhalten wir in der Variablen VAR in den Zeilen 4370 bis 4390. Je nach horizontaler Position des Cursors muß nun die Variable VAR mit 1, 4, 16 bzw. 64 multipliziert werden. Dazu benützen wir den String HILFE$. In Zeile 4400 wird schließlich das Bitmuster im Zeichensatz geändert.

In Zeile 4410 übernehmen wir das neue Bitmuster in unsere Variable für den Hintergrund, und springen nach 4000 um das ganze zu wiederholen, bis der Pfeil nach unten angewählt wird. In diesem Fall springen wir ins Menü zwei zurück (Zeile 4070).

Haben wir das Menü zwei verlassen, indem wir mit dem Cusor oben aus dem Textfenster gefahren sind, so verzweigt das Programm an die Stelle 4430. Als Zeichen haben wir nun das Leerzeichen (Zeile 4440). Dies wandeln wir mit Hilfe des Unterprogramms INTOAS in den entsprechenden ATASCII-Wert um. Dann löschen wir die Matrix, setzen die Koordinaten zum Zeichnen und lassen den Cursor im Textfenster verschwinden.

Da der Cursor im oberen Teil des Bildschirms normal immer als das Zeichen dargestellt wird, das gerade in Bearbeitung ist, ergibt sich jetzt das Problem, daß Cursor und Hintergrund dasselbe Zeichen sind, nämlich das Leerzeichen. Dieser Fall kann auch auftreten, wenn wir mit einem Zeichen auf dasselbe Zeichen schreiben wollen. In diesen Fällen wird in dem Unterprogramm BLINK das Zeichen im Cursor nicht mit dem Hintergrund vertauscht, sondern mit dem Zeichen mit dem Colorwert FARBE. FARBE = 3 bedeutet dabei, daß das nächste Zeichen aufgenommen wird, und FARBE = 4

bedeutet, daß das Zeichen bei Knopfdruck abgesetzt wird.

In den Zeilen 4510 bis 4560 wird der Cursor solange bewegt, bis entweder das Menü 2 verlassen wird, oder mit Knopfdruck ein Zeichen aufgenommen wird. Im letzten Fall schreiben wir die Koordinaten X und Y in die Variablen XALT und YALT, um diese Werte als aktuelle Zeichenposition zu behalten. In den Zeilen 4600 bis 4630 wandeln wir das Zeichen in den ATASCII-Code um, zeichnen es in die Matrix, positionieren den Cursor im Textfenster und springen dann nach 3850, wo wir das Zeichen wieder an beliebiger Stelle absetzen können, bzw. die Kodierung ändern.

Speichern

Diese Routine ist analog zur Routine Laden, mit Ausnahme der Zeile 4700. In dieser Zeile wird mit Hilfe des Unterprogramms OUTMX die Matrix gelöscht, die wir nicht mit aufnehmen wollen.

Nur noch Zeichnen

Da wir das Textfenster ausblenden, aber den Zeichensatz und den Bildschirm nachher aufnehmen wollen, müssen wir zuerst den Dateinamen eingeben. Dies geschieht in dem Unterprogramm IOMENU. Danach wird die Matrix gelöscht, und auch das Textfenster mit dem Befehl GRAPHICS 60. Da bei GRAPHICS-Befehlen grundsätzlich CHBAS den Wert 224 erhält und alle Farben ihre Ausgangswerte bekommen, müssen wir diese Register wieder mit unseren Werten laden (Zeile 4920 bis 4950).

Die Schleife von 4980 bis 5120 dient zum Kopieren einzelner Zeichen. Dabei werden die Zeilen 4990 bis 5010 solange durchlaufen bis mit Knopfdruck ein Zeichen aufgenommen wurde. Dieses Zeichen wird dann in die Variable ZA übernommen, anschließend wird solange gewartet, bis der Knopf wieder losgelassen wird, und dann die Zeilen 5060 bis 5080 solange durchlaufen, bis das Zeichen wieder abgesetzt wird. In Zeile 5120 springen wir zu Zeile 4980 und können das nächste Zeichen aufnehmen.

Diese Routine wird erst verlassen, wenn in dem Unterprogramm COPYZ die Starttaste gedrückt wurde. Die Abfrage für die START-,SELECT- und OPTION-Tasten gehen alle über das Register CONSOL (Adresse 53279). Dabei erhält man folgende Werte:

3 - OPTION-Taste

5 - SELECT-Taste
6 - START-Taste

Die HELP-Taste kann auf diese Weise nicht abgefragt werden (siehe Anhang).

MENU2

In diesem Unterprogramm werden alle Zeichen - bis auf die acht am Anfang des Programmes definierten - auf den Bildschirm gebracht. Außerdem wird in die unterste Zeile das Wort MENUE, als Option für den Rücksprung ins Hauptprogramm, geschrieben. Mit den Befehlen POKE 82,0 und POKE 83,39 setzen wir den linken Rand an die 0. Stelle und den rechten Rand an die 39. Stelle. Der Befehl POKE 766,1 bewirkt, daß Steuerzeichen nicht in ihrer Funktion ausgegeben werden, sondern als Zeichen. In den Zeilen 5270 bis 5290 schreiben wir alle Zeichen hintereinander auf den Bildschirm. Anschließend schreiben wir das Wort MENUE in die letzte Zeile. In Zeile 5310 schalten wir schließlich die Funktion der Steuerzeichen wieder ein.

ZITOMX

Dieses Unterprogramm bringt das Zeichen ZI in die Matrix. Da das Zeichen in der Variablen ZI im internen Code vorliegt, kann dieser Wert - mit 8 multipliziert - als Index für den Zeichensatz verwendet werden. In den Zeilen 5360 bis 5390 wird zunächst einmal - je nachdem, ob für das Zeichen die Normal- oder Inversdarstellung gewählt wurde - das entsprechende Farbfeld eingeschaltet, und das andere gelöscht. In den Zeilen 5400 bis 5490 wird schließlich das Zeichen dargestellt.

Da die Kodierung für eine Zeile als Dezimalzahl vorliegt, und ein Pixel aus zwei Bit besteht, also die Werte null bis drei annehmen kann, müssen wir die Dezimalzahl ins Vierersystem umwandeln. Dies geschieht auf die herkömmliche Art und Weise:

1. Teile die Zahl durch vier
2. Nimm den Rest mal vier als nächste Stelle der Zahl im Vierersystem (von rechts)
3. Lasse den Rest der Teilung weg (INT)
4. Gehe mit dieser Zahl nach eins, falls sie größer als null ist, sonst fertig.

Diese Umwandlung geschieht im Programm in den Zeilen 5400 bis 5490.

MOVEZ

In diesem Unterprogramm wird der Joystick abgefragt, und die Koordinaten werden entsprechend geändert. Dabei wird darauf geachtet, ob der X-Wert kleiner als 39 bzw. größer als 0 ist, und ob der Y-Wert größer als 0 ist. Da die Höhe des Bildschirms variiert, gehen wir von der obersten Position nicht nach unten.

BLINK

Dieses kurze Unterprogramm setzt den Cursor an die Stelle, die durch die Variablen X und Y gegeben ist. Die Farbe des Cursors steht dabei in der Variablen ZA. In Zeile 5810 wird die Abfrage durchgeführt, ob Cursor und Hintergrundzeichen übereinstimmen. Wenn ja, wird als Zeichen das in der Variablen FARBE stehende genommen.

COPYZ

Das Unterprogramm COPYZ schreibt in Zeile 5880 zunächst das Hintergrundzeichen an die Stelle X,Y. Sodann wird gefragt, ob die START-Taste gedrückt wurde. Wenn ja, springen wir zu Zeile 4710 um alles zu speichern. Dadurch umgehen wir das RETURN in Zeile 5930, und müssen demnach einen POP-Befehl setzen. Wurde die START-Taste nicht gedrückt, so gehen wir in das Unterprogramm MOVEZ, das die Werte X und Y aktualisiert. Da wir kein Textfenster mehr haben, kann die Variable Y Werte bis 23 annehmen (Zeile 5910). In Zeile 5920 rufen wir das Unterprogramm BLINK auf, das den Cursor wieder setzt.

INTOAS

Dieses Unterprogramm verwandelt den internen Code eines Zeichens, das in der Variablen ZI steht, in den ATASCII-Code, der in der Variablen ZA abgespeichert wird. Diese Umwandlung funktioniert nur für interne Codes kleiner 128. Da die Variable ZI nie andere Werte annimmt, genügt uns diese Kurzform.

OUTMX (6010 bis 6080)

Das Unterprogramm OUTMX wird bei den Optionen ENDE und TOTAL verwendet, um die gesamte Matrix mit Umrandung und den Farbfeldern zu löschen.

ASTOIN

Dieses Programm wandelt den ATASCII-Code eines Zeichens in der Variablen ZA in den internen Code um, der in die Variable ZI geschrieben wird. Da die Variabel ZI nur Werte zwischen 0 und 127 annehmen soll (d.h. nur den Code für normale Carakter), wird zunächst in Zeile 6130 geprüft, ob das Zeichen in ZA invers dargestellt ist. Wenn ja, ziehen wir 128 ab, um das entsprechende normale Zeichen zu bekommen.

POSZA

Hier wird die Position des Zeichens ZA in Menü 2 bestimmt. Da in diesem Menü die ersten acht Zeichen nicht dargestellt sind, ziehen wir von der Variablen ZA zuerst 8 ab (Zeile 6210). In Zeile 6220 wird gegebenenfalls 128 abgezogen, falls das Zeichen in ZA invers dargestellt ist. Da wir - wie oben erwähnt - eine Position links des Zeichens anwählen müssen, müssen wir beachten, ob die Spalte des Zeichens die Null ist. In Zeile 6250 wird in diesem Fall die Spalte 39 angewählt.

IOERR

Dieses Unterprogramm wird immer angesprungen, wenn ein Fehler beim Laden bzw. Speichern vorgekommen ist. Die Fehlernummer steht dabei im Register ERRSAVE (Adresse 195). In Zeile 6320 fragen wir ab, ob dies ein Fehler der Art 'Gerät nicht gefunden' war. Dieser Fall tritt z.B. ein, wenn man nur die RETURN-Taste drückt, aber auch wenn man das 'D:' für den Zugriff auf die Diskette vergißt. Wurde aber z.B. beim Laden das angegebene File nicht gefunden, so geben wir in Zeile 6330 die Nummer des Fehlers aus und warten bis eine Taste gedrückt wird zum weitermachen.

IOMENO

Zunächst löschen wir in Zeile 6400 den Tastaturpuffer. In Zeile 6410 schalten wir den Cursor ein, weil das die Eingabe erleichtert. In Zeile 6440 lesen wir den Filenamen ein und schalten dann den Cursor wieder aus. Zu bemerken ist noch, daß ein reines RETURN bei einem INPUT noch keinen Fehler verursacht. Der String ist in diesem Fall leer.

MENU1

Zunächst schalten wir den Cursor aus, setzen den linken Rand auf 4 und den rechten Rand auf 39. Damit ist eine Bildschirmzeile 36=9x4 Zeichen groß, was uns den Übergang von einer Option auf die andere über den Bildschirmrand hinweg wesentlich erleichtert.

OPNORM

Dies ist eine Zusammenfassung aller Optionen in Normalschrift. Bei den Optionen von Menü 1, wird dabei der Cursor auf den ersten Buchstaben zurückgesetzt. Dies ist vorteilhaft, weil dann die nächste Option neun Zeichen rechts liegt, und die vorhergehende neun Zeichen links.

OPINV

Analog OPNORM für inverse Optionen.

CHGET

Diese Routine simuliert die Werte, die man durch ST=STICK (0) erhält. Ein GET allein würde nicht weiterhelfen, da der Computer sonst warten würde, bis eine Taste gedrückt wird. Deshalb fragen wir vorher in Zeile 6870 ab, ob überhaupt eine Taste gedrückt wurde und übernehmen dann erst die Taste in die Variable TASTE.

KNOPF

Hier wird erfragt, ob die SELECT-Taste gedrückt ist. Um den Befehl TR=STRIG(0) zu simulieren, der bei gedrücktem Trigger den Wert 0 liefert, muß hier das Ungleichheitszeichen stehen.

Wer einen Joystick besitzt kann das Programm bedienungsfreundlicher und schneller gestalten, indem man folgende Zeilen ändert:

statt 'GOSUB CHGET' - 'ST=STICK(0)

in den Zeilen 1900, 2310, 3320, 3640, 4000 und 5540

statt 'GOSUB KNOPF' - 'TR-(PEEK(53279)=6)'

in den Zeilen 2080, 2580, 3530, 3610, 3770, 3920, 4340, 4550, 5030, 5070 und 5100

Variablenübersicht

ZNEU **1000 - 6990**

Variablen:

Name	Bedeutung
ALTZA	Zeichen unter dem Cursor auf dem Bild (ATASCII)
BYTE	Byte und Zweierpotenz (4360)
COL4	Farbe von Farbfeld 4 (3590-3800)
COL5	Farbe von Farbfeld 5 (3590-3800)
CURSOR	Farbe des Cursor in der Matrix
FARBE	Farbe, die gerade geändert wird ('Farben ändern')-Cursorfarbe, falls Vorder-und Hintergrundzeichen gleich sind ('BLINK')
FARBREG	aktuelles Farbregister ('Farben ändern')
PIXEL	Farbe des Zeichens unter dem Cursor in der Matrix
SCHLEIFE	Schleifenvariable
SPALTE	aktuelle Y-Position des Cursors in Menü 2
ST	Joystickwert, bzw. Äquivalent bei Tastatursteuerung
TASTE	ATASCII-Wert der gedrückten Taste
TR	= 0, falls Trigger bzw. SELECT gedrückt, sonst 1
VAR	Hilfsvariable
WAHL	Nummer der gewählten Option in Menü 1
WAIT	Anzahl der Schleifendurchläufe bis zum nächsten Blinken
X	aktuelle X-Koordinate (Bild)
XALT	letzte X-Koordinate, an der ein Zeichen aufgenommen oder abgesetzt wurde (Bild)
XMX	X-Koordinate (Matrix)
Y	aktuelle Y-Koordinate
YALT	letzte Y-Koordinate, an der ein Zeichen auf-

	genommen oder abgesetzt wurde (Bild)
YMX	Y-Koordinate (Matrix)
ZA	aktuelles Zeichen (ATASCII)
ZADR	Adresse des Zeichens im Zeichensatz
ZI	aktuelles Zeichen (interner Code)
ZSATZ	Adresse des Zeichensatzes

Zeichenreihen / Strings:

Name	Länge	Bedeutung
COPY$	(33)	Maschinenprogramm zum Copieren des Zeichensatzes
FARBEN$	(5)	aktuelle Farbwerte
FNAME$	(14)	Filename
HILFE$	(8)	Hilfsstring (s. Text)
MORI$	(128)	gibt für jedes Zeichen an, ob es normal oder invers dargestellt werden soll

Konstanten (Zeilennummern der Unterprogramme)

Name	Nummer	Bedeutung
ASTOIN	6120	verwandelt ATASCII in internen Code
BLINK	5800	Blinkroutine
CHGET	6870	liest Zeichen von der Tastatur
COPYZ	5880	kopiert ein Zeichen
INTOAS	5970	verwandelt internen Code in ATASCII
IOERR	6310	behandelt I/O-Fehler
IOMENU	6400	Menü für Ein- und Ausgaben
KNOPF	6980	liefert Wert vom Trigger/SELECT-Taste
MENU1	6500	zeichnet Menü 1
MENU2	5240	zeichnet Menü 2
MOVEZ	5540	bewegt aktuelles Zeichen
MXCLR	5700	löscht Zeichen in der Matrix
OPINV	6730	zeichnet inverse Option
OPNORM	6590	zeichnet normale Option
OUTMX	6040	löscht Matrix
POSZA	6210	bestimmt die Position des aktuellen Zeichens in Menü 2
ZITOMX	5360	zeichnet aktuelles Zeichen in die Matrix

Unterprogrammaufrufe :

in	nach	Zweck
1900	CHGET	Joystick lesen zur Postenauswahl
1920	OPNORM	Option in Normalschrift
2070	OPINV	Option in inverser Schrift
2080	KNOPF	Trigger gedrückt ?
2100	OPNORM	Option in Normalschrift
2120	und 2130	bedingter Unterprogrammaufruf
2140	MENU1	Menü wird nach RETURN neu gezeichnet
2200	OPINV	alte Option wird invertiert
2290	MXCLR	Matrix löschen
2310	CHGET	Joystickwert holen
2580	KNOPF	Trigger gedrückt ?
2670	IOMENU	Filenamen eingeben
2900	IOERR	Fehlerbehandlung
3080	ZITOMX	Zeichen in die Matrix bringen (evtl. Leerzeichen
3090	MENU2	Menü 2 aufrufen (s. Text)
3100	POSZA	Position des Zeichens in Menü 2
3250	MENU2	Menü 2 zeichnen
3320	CHGET	Joystickwert holen
3400	OPINV	'MENUE' invertieren
3420	OPNORM	'MENUE' normal schreiben
3490	OPINV	'MENUE' invertieren
3530	KNOPF	Trigger gedrückt ?
3580	INTOAS	verwandelt das aufgenommene Zeichen in ATASCII-Code
3610	KNOPF	Warten bis Knopf losgelassen wird
3640	CHGET	Joystickwert holen
3770	KNOPF	Trigger gedrückt ?
3820	ZITOMX	aktuelles Zeichen in die Matrix
3880	MOVEZ	Cursor-Koordinaten aktualisieren
3910	BLINK	zeichnet Cursor
3920	KNOPF	Trigger gedrückt ?
4000	CHGET	Joystickwert holen
4340	KNOPF	Trigger gedrückt ?
4450	INTOAS	verwandelt das Leerzeichen in ATASCII-Code
4460	MXCLR	Matrix löschen
4520	MOVEZ	Cursor-Koordinaten aktualisieren
4540	BLINK	Cursor zeichnen
4550	KNOPF	Trigger gedrückt ?
4600	ASTOIN	verwandelt aufgenommenes Zeichen in internen Code
4610	ZITOMX	bringt aktuelles Zeichen in die Matrix

4620	POSZA	bestimmt die Position des aktuellen Zeichens in Menü 2
4670	IOMENU	Filenamen einlesen
4700	OUTMX	Matrix löschen
4830	IOERR	Fehlerbehandlung
4880	IOMENU	Filenamen einlesen
4910	OUTMX	Matrix löschen
4990	COPYZ	Zeichen kopieren
5000	KNOPF	Trigger gedrückt ?
5030	KNOPF	Trigger gedrückt ?
5060	COPYZ	Zeichen kopieren
5070	KNOPF	Trigger gedrückt ?
5100	KNOPF	Trigger gedrückt ?
5140	IOERR	Fehlerbehandlung
5300	OPNORM	schreibt 'MENUE'
5540	CHGET	Joystickwert holen
5560 - 5610		Teil der Bewegungsroutine
5900	MOVEZ	Cursor Koordinaten aktualisieren
5920	BLINK	Cursor zeichnen

Ein weiterer Textmodus

Es gibt noch einen fünften Textmodus, doch leider ist dieser nicht vom Basic aus aufrufbar. Dies ist ein Textmodus, indem die Zeichen acht Punkte breit und zehn Punkte hoch sind, so daß Buchstaben mit Unterlängen definiert werden können. Um diesen Mode zu definieren, müssen wir uns zunächst die Display List (das ist das Programm für den Video Baustein) betrachten (Die Display List wird erst in Kapitel 2.3 näher untersucht, wir geben hier nur der Vollständigkeit halber ein Programm an, das Buchstaben mit Unterlängen kreiert. Wer die Display List nicht kennt, sollte vorher Kapitel 2.3 lesen.)

In Adresse 560, 561 finden wir den Zeiger SDLSTL auf den Beginn dieses Programmes. Läßt man sich in GRAPHICS 0 ab dieser Adresse 32 Zeichen ausgeben, so erhält man folgendes (64K-Version / normale Display List):

```
112   8 Leerzeilen   (für              )
112   8 Leerzeilen   (verzerrungsfreien)
112   8 Leerzeilen   (Bildschirm       )
 66   Grafik-0-Zeile mit der Option
 64      Bildschirmspeicheranfangsadresse
156      laden (156*256+64=40000)
  2   Grafik-0-Zeile
```

```
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
  2   Grafik-0-Zeile
 65   Rücksprungbefehl nach 156*256+32=39968
 32      Adresse der
156      Display List
```

Die Zahl 112 am Anfang bedeutet jeweils acht leere (Fernseh-) Zeilen. Die 66 muß man als 64 + 2 betrachten, wobei die 2 der Befehl für eine GRAPHICS 0-Zeile ist, und die 64 angibt, daß die nächsten beiden Bytes die Adresse für den Bildschirm sind. Nun folgt noch 23 mal '2' für die restlichen 23 Grafik-0-Zeilen. Die folgende Zahl 63 ist ein Sprungbefehl auf eine Adresse, die durch die nächsten beiden Byte angegeben wird (das erste Byte ist jeweils das LSB, das zweite Byte immer das MSB). Die dort angegebenen Adresse ist der Anfang der Display List. Diese Endlosschleife gibt also für jede Zeile einen bestimmten Grafik-Mode an. Mit der Befehlsfolge:

```
POKE PEEK(560) + 256 * PEEK(561) + 6,3
```

erzeugen wir eine Zeile, die zehn Pixel hoch ist (2. Zeile von oben). Da alle Zeichen nur acht Punkte hoch sind, bleiben die unteren beiden Fernsehzeilen dieser Zeile in der Regel frei. Lediglich die Zeichen aus dem letzten Viertel des Zeichensatzes werden so dargestellt, daß die oberen beiden Fernsehzeilen leer bleiben, und die ersten zwei Byte der Kodierung dieses Zeichens in den unteren beiden Fernsehzeilen dargestellt werden. Dadurch ist eine

Darstellung von (Klein-)Buchstaben mit Unterlängen möglich.

Da alle Buchstaben normalerweise so definiert sind, daß sowohl die oberste als auch die unterste Zeile frei bleiben, sitzen alle Buchstaben innerhalb dieser zehn Fernsehzeilen etwas zu hoch. Demnach sollte man beim Kopieren des Zeichensatzes vom ROM ins RAM alle Zeichen um ein Punkt nach unten verschieben. Da dies in Basic eine etwas langwierige Angelegenheit ist, geben wir dazu ein Maschinenprogramm an, das vom Basic aus aufgerufen werden kann. Außerdem wird in diesem Programm der Zeichensatz etwas verändert, so daß einige Buchstaben Unterlängen bekommen.

```
1000 REM ***************************************************
1010 REM *          BUCHSTABEN MIT UNTERLAENGEN           *
1020 REM ***************************************************
1030 REM
1040 REM
1050 REM ***************************************************
1060 REM *                     VORSPANN                   *
1070 REM ***************************************************
1080 DIM COPY$(52)
1090 POKE 106,PEEK(106)-8:GRAPHICS 0
2000 REM ***************************************************
2010 REM *                NEUE DISPLAY LIST                *
2020 REM ***************************************************
2030 POKE 559,0
2040 DLIST=PEEK(560)+256*PEEK(561)
2050 POKE DLIST+3,67
2060 FOR SCHLEIFE=6 TO 25
2070 POKE DLIST+SCHLEIFE,3
2080 NEXT SCHLEIFE
2090 POKE DLIST+26,65
2100 POKE DLIST+27,PEEK(560)
2110 POKE DLIST+28,PEEK(561)
2120 POKE 559,34
3000 REM ***************************************************
3010 REM *        KOPIERE UND AENDERE ZEICHENSATZ          *
3020 REM ***************************************************
3030 FOR SCHLEIFE=1 TO 52
3040 READ BYTE:COPY$(SCHLEIFE)=CHR$(BYTE)
3050 NEXT SCHLEIFE
3060 DATA 104,104,133,209,104,133,208,133,206,169
3070 DATA 224,133,207,160,0,152,145,208,177,206
3080 DATA 200,145,208,192,7,208,247,165,208,24
3090 DATA 105,8,133,208,133,206,144,231,230,209
3100 DATA 165,207,24,105,1,133,207,201,228,208,218,96
3110 ZSATZ=256*(PEEK(106)+4)
```

```
3120 DUMMY=USR(ADR(COPY$),ZSATZ)
3130 FOR SCHLEIFE=1 TO 6
3140 READ ZNUM,BYTE1,BYTE2
3150 POKE ZSATZ+8*ZNUM,BYTE1
3160 POKE ZSATZ+8*ZNUM+1,BYTE2
3170 NEXT SCHLEIFE
3180 DATA 102,24,24,103,70,60,106,6,60,112,96,96
3190 DATA 113,6,6,121,56,0
3200 POKE 756,PEEK(106)+4
```

Da wir unser Maschinenprogramm in einen String schreiben (dies wird später noch genauer erläutert), erzeugen wir dafür in Zeile 1080 den 52-Zeichen langen String COPY$. In Zeile 1090 schaffen wir 8x256 Zeichen Platz für unseren neuen Zeichensatz.

In den Zeilen 2000 bis 2120 schreiben wir eine neue Display-List für den Textmode mit Unterlängen. Der Befehl Poke 559,0 in Zeile 2030 schaltet den Videoprozessor aus, damit er nicht 'durcheinandergerät', wenn wir die neue Display-List schreiben. Wir ersetzen in den Zeilen 2050 bis 2080 jede 2 der GRAPHICS 0 Display-List durch eine 3. Allerdings können wir nicht wie üblich 24 Zeilen darstellen, da jetzt jede Zeile 10 Pixel hoch ist, und wir auf dem Monitor höchstens 192 Pixel senkrecht darstellen können. In Zeile 2090 steht der Rücksprungbefehl mit Angabe des Anfangs der Display-List in den Zeilen 2100 und 2110. In Zeile 2120 schalten wir den Videoprozessor wieder ein.

Ab Zeile 3030 wird das Maschinenprogramm eingelesen. In Zeile 3120 rufen wir das Maschinenprogramm auf, wobei wir dem Maschinenprogramm die neue Adresse unseres Zeichensatzes angeben müssen (ZSATZ).

In den Zeilen 3130 bis 3180 verändern wir die Zeichen f, g, j, p, q, y. Dabei lesen wir zuerst die Nummer des Zeichens ein, und dann die beiden Byte, die für die Unterlänge verantwortlich sind.

In Zeile 3190 schalten wir schließlich auf den neuen Zeichensatz um.

2.2 Grafikmodes

Es stehen uns elf Grafikmodes zur Verfügung, die alle vom Basic aus aufrufbar sind. Bei allen, bis auf den drei sogenannten GTIA-Modes (9,10, und 11), können wir dabei

wahlweise ein Textfenster einblenden. Dabei ist zu bemerken, daß sich der Speicherbereich des Textfensters nicht mit dem Bereich der Grafik überlagert, wie man in folgendem Beispiel sieht.

```
1000 GRAPHICS 5
1010 COLOR 1:PLOT 0,0:DRAWTO 79,47
1020 PRINT "TEXTFENSTER"
1030 IF PEEK(53279)<>6 THEN 1030
1040 GRAPHICS 5+16+32
1050 IF PEEK(53279)<>5 THEN 1050
1060 GRAPHICS 5+32
1070 GOTO 1030
```

Eine recht einfache Sache, die aber erfahrungsgemäß immer wieder Schwierigkeiten bereitet, ist die Handhabung der COLOR und SETCOLOR-Befehle. Nehmen wir als Beispiel den Grafikmode 3. Dies ist ein Vierfarbmode, d.h. jedes Pixel wird durch zwei Bit kodiert. Diese beiden Bits können nun die Werte 00, 01, 10 und 11 annehmen. Um einen Punkt zu zeichnen, muß man also vorher ein Bitmuster auswählen. Dies geschieht mit dem COLOR-Befehl. Nun muß noch jedem Bitmuster eine Farbe zugeordnet werden. Dies übernimmt der SETCOLOR-Befehl. Doch dabei stimmt leider der erste Wert des SETCOLOR-Befehles nicht mit dem Bitmuster (als Dezimalzahl gelesen) überein (siehe Tabelle im Anhang).

Eine Besonderheit stellen die drei GTIA-Modes dar. In diesen drei Modes wird ein Pixel durch vier Bit kodiert, d.h. ein Pixel kann 16 verschiedene Farbwerte annehmen. Da es nur neun Farbregister gibt (5 für die Grafikmodes und 4 weitere für Player-Missiles), sind die sechzehn Farben in den GTIA-Modes nicht frei wählbar.

Im Grafikmode 9 steht uns eine Farbe in sechzehn Helligkeiten zur Verfügung. Die Farbe wählen wir mit dem Befehl

SETCOLOR 4, Farbe, Helligkeit

Dabei sollte man die Helligkeit Null (= dunkel) setzen, da sich die wirkliche Helligkeit eines Punktes durch eine logische ODER-Verknüpfung des COLOR-Wertes und der in den SETCOLOR-Befehl angegebenen Helligkeit zusammensetzt. Wir werden diesen Mode verwenden, um in einem Spiel die Schattierung des Himmels darzustellen.

Der Grafikmode 11 ist dem Mode 9 sehr ähnlich. In diesem Mode können sechzehn verschiedene Farben,aber alle in der

selben Helligkeit auf den Bildschirm gebracht werden. Die Helligkeit bestimmen wir wieder mit dem Befehl

SETCOLOR 4, Farbe, Helligkeit

hier müssen wir aus Gründen analog zu oben den Farbwert Null Setzen.

```
1000 REM GRAPHIC 11 DEMO
1010 GRAPHICS 11
1020 SETCOLOR 4,0,8
1030 FOR J=0 TO 3
1040 FOR FARBE=1 TO 15
1050 COLOR FARBE
1060 PLOT 0,45*J+3*FARBE:DRAWTO 79,45*J+3*FARBE
1070 PLOT 0,45*J+3*FARBE+1:DRAWTO 79,45*J+3*FARBE+1
1080 PLOT 0,45*J+3*FARBE+2:DRAWTO 79,45*J+3*FARBE+2
1090 NEXT FARBE
1100 NEXT J
1110 GOTO 1110
```

Man erkennt, daß in diesen beiden Modes - im Unterschied zu den übrigen Grafikmodes - der Wert im COLOR-Befehl nicht ein Index für ein Farbregister ist, sondern die Farbe, bzw. Helligkeit eines Punktes selbst bestimmt.

Im Grafikmode 10 stehen uns neun frei wählbare Farben zur Verfügung. Obwohl jeder Punkt mit vier Bit kodiert ist stehen uns nur neun Farben zur Verfuegung, da es nur neun Farbregister gibt. Hierbei ist der Wert im COLOR-Befehl wieder als Index auf eines der Farbregister zu verstehen. Da mit dem SETCOLOR-Befehl nur fünf Farbregister angesprochen werden können, müssen die restlichen Farbwerte mit dem POKE-Befehl direkt in die entsprechenden Register geschrieben werden. Die Farbregister haben die Adresse 704 bis 712, wobei die Register 704 bis 707 für die Farben der Player-Missile verantwortlich sind. Im Grafikmode 10 brauchen wir diese Register nun auch für unsere Farbpunkte. Die Register 708 bis 712 können sowohl durch einen POKE-Befehl, wie auch mit dem SETCOLOR-Befehl voll geschrieben werden. Dabei entsprechen sich die Werte folgendermaßen:

Erster Wert bei SETCOLOR	Register
0	708
1	709
2	710
3	711
4	712

Der Wert, der in die Register hineinzuschreiben ist, ergibt sich aus 16 * Farbe + Helligkeit.

SETCOLOR A,B,C entspricht also POKE 708+A,16*B+C

Das Zeichnen im Grafikmode 10 funktioniert somit folgendermaßen:

1. Mit dem POKE- bzw. SETCOLOR- Befehl schreiben wir die Farbregister voll.
2. Mit dem COLOR-Befehl wählen wir einen Index auf eines dieser Farbregister.
3. Nun kann ganz normal mit dem PLOT- und dem DRAWTO-Befehl gearbeitet werden.

Außer den Befehlen zum Zeichnen von Punkten und Linien gibt es noch einen Befehl zum Ausfüllen von Flächen. Dieser Befehl ist in der Basic-Begleitbroschüre von ATARI, die wir erhalten haben, nicht ausgeführt, deshalb behandeln wir ihn etwas ausführlicher. Dieser Befehl ist ein allgemeiner Befehl, der Betriebssystemkommandos simuliert. Der Befehl zum Ausfüllen von Flächen lautet:

```
XIO 18, #6, 12, 0, "S:"
```

Dabei muß die Farbe, mit der die Fläche ausgefüllt werden soll, in das Register FILDAT (Adresse 765) geschrieben werden (die anderen Möglichkeiten die der XIO-Befehl bietet, werden wir in Kapitel 4 betrachten).

Um zu sehen, wie der XIO-FILL Befehl arbeitet, betrachten wir folgendes kleines Programm:

```
1000 GRAPHICS 5
1010 COLOR 1
1020 PLOT 40,0
1030 DRAWTO 45,3
1040 DRAWTO 20,19
1050 PLOT 70,30
1060 DRAWTO 79,39
```

```
1070 DRAWTO 0,35
1080 POSITION 0,2
1090 POKE 765,1
1100 FOR I=1 TO 100:NEXT I
1110 XIO 18,#6,12,0,"S:"
```

Startet man das Programm mit RUN, so erkennt man, daß die Fläche durch Zeichnen von waagrechten Linien gefüllt wird. Eine waagrechte Linie wird dabei solange gezeichnet, bis sie an einen Punkt gelangt der schon gesetzt ist. Die linke Begrenzung der Fläche ergibt sich aus einer Linie von der letzten Cursor-Position zu der aktuellen. Also können mit dieser Methode Flächen gezeichnet werden, die oben und unten durch eine waagerechte, auf der linken Seite durch eine gerade (nicht notwendigerweise senkrechten) Linie begrenzt werden. Die rechte Begrenzung ist frei wählbar. Das bedeutet aber auch, daß nur auf den Hintergrund gezeichnet werden kann.

2.3 Texte und Grafiken mischen

Vom Basic aus kann man mit dem GRAPHICS-Befehl 16 verschiedene Grafikmodes aufrufen. Außerdem gibt es in jedem Mode, bis auf die GTIA-Modes (9, 10 und 11) die Möglichkeit ein Textfenster einzublenden. Dies bedeutet eine horizontale Aufspaltung des Bildschirms in zwei verschiedene Grafikmodes. In Kapitel 2.1 hatten wir auch schon zwei Grafikmodes gemischt, als die zweite Grafik-0-Zeile in eine Zeile für Buchstaben mit Unterlängen umgewandelt wurde. Dies deutet schon darauf hin, daß man nicht unbedingt auf die vom Basic aus aufrufbaren Grafikmodes gebunden ist.

In der Tat kann man auf dem Fernseher beliebig viele Grafikmodes miteinander mischen, wobei jedoch für eine ganze Zeile der Grafikmode festliegt. Um dem Video-Prozessor mitzuteilen, in welcher Zeile welcher Grafikmode darzustellen ist, muß es eine Liste geben, die die Reihenfolge der Grafikmodes (zeilenweise von oben nach unten) angibt. Diese Liste heißt **DISPLAY LIST**, die als Programm für den Videoprozessor zu interpretieren ist. Außer den Befehlen für die verschiedenen Grafikmodes enthält die Display List noch gewisse andere Befehle. Ein Teil des Befehlsatzes ist im folgenden angegeben:

Grafik-Mode Videoprozessor	entsprechender Basic-Mode	Erklärung
2	0	Text-Mode
3	-	Text-Mode
4	12	Text-Mode
5	13	Text-Mode
6	1	Text-Mode
7	2	Grafik-Mode
8	3	Grafik-Mode
9	4	Grafik-Mode
10	5	Grafik-Mode
11	6	Grafik-Mode
12	14	Grafik-Mode
13	7	Grafik-Mode
14	15	Grafik-Mode
15	8	Grafik-Mode
+64		Bildschirm-adresse laden
112		8 Leerzeilen
65		Sprungbefehl

Wie man sieht, stimmen dabei die Nummern der Basic-Modes nicht mit den entsprechenden Nummern der Kommandos für den Videoprozessor überein, so daß man beim Erstellen einer eigenen Display List die 'gewohnten' Werte für Grafikmodes immer anhand der Tabelle in die entsprechenden Werte umrechnen muß.

Außer der Aufteilung des Bildschirms in die verschiedenen Grafikmodes, muß der Videoprozessor auch noch wissen, wo er die Informationen für den Bildinhalt aus dem RAM herauszulesen hat. Dazu dient die Option '+64', wobei die beiden nächstfolgenden Byte die Adresse angeben (in der Form LSB, MSB).

Als einfachstes Beispiel haben wir schon die GRAPHICS-0-Display List angegeben. Wie in jeder Display List steht am Anfang 3x112, für 3x8 Leerzeilen. Dies ist offensichtlich nicht unbedingt notwendig, und man kann deshalb versuchen gleich mit einem Kommando für einen Grafikmode zu beginnen, wobei es allerdings passieren kann, daß der Bildschirm 'umkippt'. Da 24 GRAPHICS-0-Zeilen dargestellt werden sollen, ist dann das erste Kommando die 66=2+64, für die Option 'Bildschirmspeicher Anfangsadresse' laden. Die nächsten beiden Byte geben diese Adresse an. Nun folgen 23 2er für die restlichen 23 GRAPHICS-0-Zeilen. Um dem Video-Prozessor anzugeben, daß die Display List hiermit zu Ende ist, folgt als letztes der Sprungbefehl an den Anfang der Display List. Die Anfangsadresse der Display List steht dabei in den auf den Sprungbefehl folgenden beiden Byte.

Auf diese Art und Weise kann man sich eine eigene Display List zusammenstellen, wobei allerdings einiges zu beachten ist. Deshalb gehen wir schrittweise vor und bringen dazu als Beispiel ein Spiel, das sich durch alle drei Bände hindurchziehen wird, und das wir schrittweise ergänzen und verfeinern werden.

Der Sinn des Spiel soll darin bestehen, Verletzte aus einem Bergwerk an die Oberfläche zu bringen. Da unser Bildschirm eine Schnittzeichnung des Bergwerkes darstellen soll, beginnen wir ganz oben im Grafikmode 9 einen Himmel zu zeichnen, darunter folgt im Grafikmode 1 die Erdoberfläche und das Bergwerk und ganz unten eine Grafik-0-Zeile, die uns den aktuellen Punktestand etc. zeigt. Die erste Schwierigkeit besteht nun darin, daß es für den Grafikmode 9 offensichtlich kein entsprechendes Kommando für den Videoprozessor gibt. Das liegt daran, daß die GTIA-Modes (9, 10 und 11) dieselbe Display List haben, wie der Grafikmode 8. Wie unterscheidet nun der Videoprozessor zwischen den Grafikmodes 8 bis 11? Dazu dient das Register GPRIOR (Adresse 623), das auch noch andere Aufgaben erfüllt, wie wir bei der Besprechung der Player Missiles noch sehen werden. Zum Umschalten von Grafikmode 8 auf die GTIA-Modes dienen dabei die höchsten beiden Bit dieses Registers. Die Zustände dieser beiden Bit entsprechen dabei in folgender Weise den Grafikmodes:

00xxxxxx	dezimal	0:	Grafikmode 8
01xxxxxx	dezimal	64:	Grafikmode 9
10xxxxxx	dezimal	128:	Grafikmode 10
11xxxxxx	dezimal	192:	Grafikmode 11

Dieses Register bewirkt also, daß der Videoprozessor die Daten im Bildschirmspeicher anders interpretiert. Diese andere Interpretation gilt nicht nur für den Grafikmode 8, sondern auch für alle anderen Modes. Dies ist der Grund warum man den Grafikmode 9 bis 11 kein Textfenster einblenden kann. Als Beispiel können wir in Grafikmode 0 den Befehl POKE 623, 64 eingeben. Nun ist die Schrift fast nicht mehr lesbar, was daran liegt, daß der Viedoprozessor die Bitmuster für die Buchstaben anders interpretiert. Unsere Mittel reichen leider noch nicht aus, um dafür eine Lösung anzugeben. Deshalb werden wir den Grafikmode 9 für den Himmel zunächst weglassen und dafür Grafik 8 verwenden. In Band 2 werden wir auch sehen, wie man in die GTIA-Modes Textfenster einblenden kann.

Beginnen wir nun schrittweise eine eigene Display List zu erstellen:

Erster Schritt:

Da das Betriebssystem streng zwischen dem Grafikmode 0 (in dem der Editor arbeitet) und den übrigen Modes unterscheidet, müssen wir ihm deshalb mitteilen, daß jetzt auch ein anderer Mode vorliegt. Dies geht am einfachsten mit einem GRAPHICS-8-Befehl. Dies ist auch die erste Zeile in unserem Beispiel, zuvor jedoch die üblichen Bemerkungen:

```
2000 REM ************************************************
2010 REM *                NEUE DISPLAY LIST             *
2020 REM ************************************************
2030 GRAPHICS 8
```

Zweiter Schritt:

Da wir das Programm für den Video-Prozessor ändern, müssen wir diesen zunächst abschalten. Dazu dient

```
2040 POKE 559,0
```

Dritter Schritt:

Nun muß man sich überlegen, wie die Aufteilung des Bildschirmes in die verschiedenen Grafikmodes erfolgen soll. Dabei ist zu beachten, daß das Bild nicht höher als 192 Fernsehzeilen (scan lines) sein darf. Da die Höhe der Pixel in den verschiedenen Grafikmodes unterschiedlich ist, entnimmt man die entsprechenden Werte aus der Tabelle im Anhang. Es sei noch bemerkt, daß es möglich ist, bis zu 200 scan lines auf den Bildschirm zu bringen, ohne daß das Bild 'umkippt'- es können dabei jedoch in den Ecken des Fernsehers erhebliche Verzerrungen auftreten; dies sollte man einmal ausprobieren.

In dem Spiel soll der Bildschirm wie folgt aufgeteilt werden:

Grafik-Mode (Basic)	Anzahl Zeilen	scan-lines pro Zeile	ergibt
8	56	1	56
1	16	8	128
0	1	8	8

		Gesamtanzahl der scan lines:	192

4. Schritt:

Nun benötigen wir die Größe des Bildschirmspeichers. Dazu liest man aus derselben Tabelle heraus, wieviel RAM jede Grafikzeile benötigt. Die Größe des Bildschirmspeichers ergibt sich dann aus der Summe dieser Werte.

In unserem Beispiel ergibt sich:

Grafik-Mode (Basic)	Anzahl Zeilen	Byte pro Zeile	ergibt
8	56	40	2240
1	16	20	320
0	1	40	40

		Gesamtbedarf an Speicherplatz:	2600

Da wir den Bildschirm später vielleicht noch im RAM verschieben wollen, rechnen wir nicht mit absoluten Adressen, sondern gehen von dem momentanen Wert in RAMTOP (Adresse 106) aus.

```
2050 RAMEND=256*PEEK(106)
2060 BILDSCH=RAMEND-2600
```

5. Schritt:

Nun benötigen wir die Länge der Display List. Diese ergibt sich zunächst aus der Anzahl der verschiedenen Grafikmode-Zeilen auf dem Fernseher plus einige Steuerkommandos für den Videoprozessor. Dazu zählen die 3x112 für die 3x8 Leerzeilen, die zwei Byte für die Bildschirmanfangsadresse und drei Byte für das Rücksprungkommando zum Anfang der Display List. Damit ergibt sich die Länge der Display List als die Anzahl der Grafikmodezeilen + 8.

Dabei sind zwei Dinge zu beachten: erstens darf der Bildschirm keine 4K-Grenze überschreiten, d.h. kein Byte des Bildschirms darf eine Adresse haben, die durch 4096 teilbar ist. Ist dies unumgänglich, wie z.B. im Grafikmode 8, so muß für die zweite Hälfte des Bildschirms (ab der Zeile, wo die 4K-Grenze überschritten wird), in der Display List die Option +64 gewählt werden, und in den nächsten beiden Byte die neue Anfangsadresse der zweiten Hälfte des Bildschirms. Dies kann man sich veranschaulichen, wenn man sich eine GRAPHICS-8-Display List ausdrucken läßt.

Das zweite Hindernis ist, daß die Display List keine 1K-Grenze überschreiten darf. Dies ist ein relativ geringes

Problem, da die Display List in der Regel diese Länge nicht erreicht. Wir werden in Band 2 längere Display Lists verwenden, und dort auch eine Lösung für dieses Problem anbieten. Zunächst einmal schauen wir nur, daß wir die Display List so setzen, daß sie keine 1K-Grenze überschreitet.

In unserem Spiel ergab sich die Länge des Bildschirms zu 2600 Byte, und die höchste RAM-Adresse ist eine 4K-Grenze. Dadurch überschreitet der Bildschirm keine 4K-Grenze, insofern wir den Zeiger in Adresse 106 nicht um 8 (d.h. 2K) nach unten verschieben. Dies müssen wir uns für spätere Erweiterungen des Spiels merken. Die Länge der Display List ergibt sich dadurch zu 81, so daß die Gesamtlänge von Bildschirm und Display List 2681 ist. Dadurch überschreitet die Display List auch keine 1K-Grenze. Wir bekommen also:

```
2070 DLIST=BILDSCH-81
```

6. Schritt:

Nun kann man ab der eben errechneten Startadresse die Display List ins RAM schreiben. In unserem Spiel:

```
2080 RESTORE 2120
```

Als erstes die 3*8 Leerzeilen, und das Kommando für die erste GRAPHICS-8-Zeile mit der Option 'Bildschirmadresse laden'.

```
2090 FOR SCHLEIFE=DLIST TO DLIST+3
2100 READ BYTE:POKE SCHLEIFE,BYTE
2110 NEXT SCHLEIFE
2120 DATA 112,112,112,79
```

Die nächsten beiden Byte sind die Bildschirmanfangsadresse

```
2130 POKE DLIST+4,BILDSCH-256*INT(BILDSCH/256)
2140 POKE DLIST+5,INT(BILDSCH/256)
```

Jetzt kommen die restlichen 55 GRAPHICS-8-Zeilen

```
2150 FOR SCHLEIFE=DLIST+6 TO DLIST+60
2160 POKE SCHLEIFE,15
2170 NEXT SCHLEIFE
```

und 16 GRAPHICS-1-Zeilen

```
2180 FOR SCHLEIFE=DLIST+61 TO DLIST+76
```

```
2190 POKE SCHLEIFE,6
2200 NEXT SCHLEIFE
```

und eine GRAPHICS-0-Zeile

```
2210 POKE DLIST+77,2
```

Als letztes benötigen wir den Rücksprung zum Anfang der Display List

```
2220 POKE DLIST+78,65
2230 POKE DLIST+79,DLIST-256*INT(DLIST/256)
2240 POKE DLIST+80,INT(DLIST/256)
```

Damit steht die Display List vollständig im RAM.

7. Schritt:

Nun müssen wir dem Videoprozessor noch angeben, wo die Display List steht. Dazu dient das Register SDLSTL (Adressen 560 und 561).

```
2250 POKE 560,PEEK(DLIST+79)
2260 POKE 561,PEEK(DLIST+80)
```

8. Schritt:

Jetzt können wir den Videoprozessor wieder einschalten.

```
2270 POKE 559,34
```

Bevor wir das Programm nun mit RUN starten, müssen wir noch eine Leerschleife eingeben, um zu verhindern, daß das Programm abbricht, z.B.

```
9999 GOTO 9999
```

Bei einem Probelauf sieht man nun oben einen hellblauen Balken. Dies ist die Origrinal-Hintergrundfarbe für den Grafikmode 8. Der schwarze Teil darunter ist der Hintergrund für die Grafik 1, und die hellblaue Zeile ganz unten ist der Grafikmode 0.

Nun weiß zwar der Videoprozessor, wo die Display List ist, und dadurch auch, wo er den Bildschirm findet, aber wir müssen dem Betriebssystem noch mitteilen, wo der Bildschirm liegt. Dazu genügt es, die Bildschirmanfangsadresse in geeignete Register zu speichern. Da das Betriebssystem zwischen dem Grafikmode 0 und den restlichen Grafikmodes

unterscheidet, hat man die Möglichkeit zwei Startadressen anzugeben. Die eine schreibt man in TXTMSC (Adressen 660 und 661). Dies ist die Anfangsadresse des Textfensters. Die andere Adresse schreibt man in SAVMSC (88, 89). Dies ist die Bildschirmsanfangsadresse für die restlichen Grafikmodes. Dabei ist allerdings zu beachten, daß das Textfenster nicht mehr als vier Zeilen hoch sein darf. Weniger ist erlaubt, allerdings sollte man jegliches Scrollen im Textfenster vermeiden, falls sich hinter dem Textfenster noch ein RAM-Bereich befindet, da -wie schon erwähnt- beim Scrollen ein Teil der Daten dahinter verloren geht.

```
2280 BILD=BILDSCH+2240
2290 POKE 87,1
2300 POKE 88,BILD-256*INT(BILD/256)
2310 POKE 89,INT(BILD/256)
2320 TEXT=BILD+320
2330 POKE 660,TEXT-256*INT(TEXT/256)
2340 POKE 661,INT(TEXT/256)
```

Bevor wir diesen Programmteil verlassen, schalten wir noch den störenden Cursor aus:

```
2350 POKE 752,1
2360 RETURN
```

Daß man mit kleineren Textfenstern vorsichtig umgehen muß, zeigt sich, wenn man das Programm mit der BREAK-Taste abbricht. Was passiert? In der obersten Zeile des vermeintlich vier Zeilen hohen Textfensters wird die Meldung STOPPED AT LINE 9999 ausgegeben, darunter eine Leerzeile, die Meldung READY und dann wird der Cursor darunter gestellt.

Da das Betriebssystem aber nicht weiß, daß unser Textfenster nur eine einzige Zeile umfasst, wird die READY-Meldung in den RAM Bereich dahinter geschrieben. Dies ist natürlich auf unserem Bildschirm nicht mehr sichtbar. Da hinter unserem Bildschirm gleich das Basic beginnt, wird das READY quasi ins ROM geschrieben. Deshalb kann man jetzt zwar Tasten bedienen, aber keine Befehle eingeben. Wir können aber mit CONTROL/CLEAR den Bildschirm löschen. Ab da sieht man wieder alle eingegebenen Zeichen. Dies hat auch den Nachteil, daß man keine Fehlermeldungen lesen kann. Um dies zu vermeiden, gibt es zwei Möglichkeiten:

1. Wenn man kein Textfenster benötigt, so kann man in Zeile 2030 auch den Grafikmode 24 wählen. Dann wird nämlich bei Fehler immer zuerst der Grafikmode 0 eingeschaltet, und dann erst die Fehlermeldung ausgegeben.

2. Wann schreibt sich eine Fehler-Routine, die im Prinzip dasselbe macht. So eine Routine ist immer praktisch, wenn man mit selbsterstellter Display List arbeitet. Diese Fehler-Routine könnte z.B. folgendermaßen aussehen:

```
0     TRAP 32000
32000 TRAP 32000
32001 GRAPHICS 0
32002 PRINT "Fehler ";PEEK(195);" in Zeile ";
      PEEK(186)+256xPEEK(187)
```

Da diese Zeilen mit dem Programmablauf im eigentlichen Sinne nichts zutun haben, werden sie nach dem Verbessern des Programms wieder gelöscht.

Nun können wir mit unseren Spiel fortfahren.

Weil wir unser Spielfeld mit Bäumen, Häusern und Leitern versehen wollen, müssen wir zunächst Platz für einen eigenen Zeichensatz schaffen. Diesen legen wir in gewohnter Weise zwischen den Bildschirm und das Basic, d.h. wir müssen den Zeiger in Adresse 106 ändern, bevor wir in das Unterprogramm 'Neue Display List' springen. Wie schon in dem Programm ZNEU gehen wir dabei von Adresse 740 aus.

```
1000 REM ***********************************************
1010 REM *                  SPIEL                      *
1020 REM ***********************************************
1030 POKE 106,PEEK(740)-4
1040 GOSUB 2000
```

Als Adresse für den neuen Zeichensatz ergibt sich damit

```
1050 ZSATZ=256*PEEK(106)
```

Das Zeichnen des Bildschirms machen wir in einem Unterprogramm ab Zeile 2500:

```
1060 GOSUB 2500
2500 REM ***********************************************
2510 REM *          SPIELFELD ZEICHNEN                 *
2520 REM ***********************************************
```

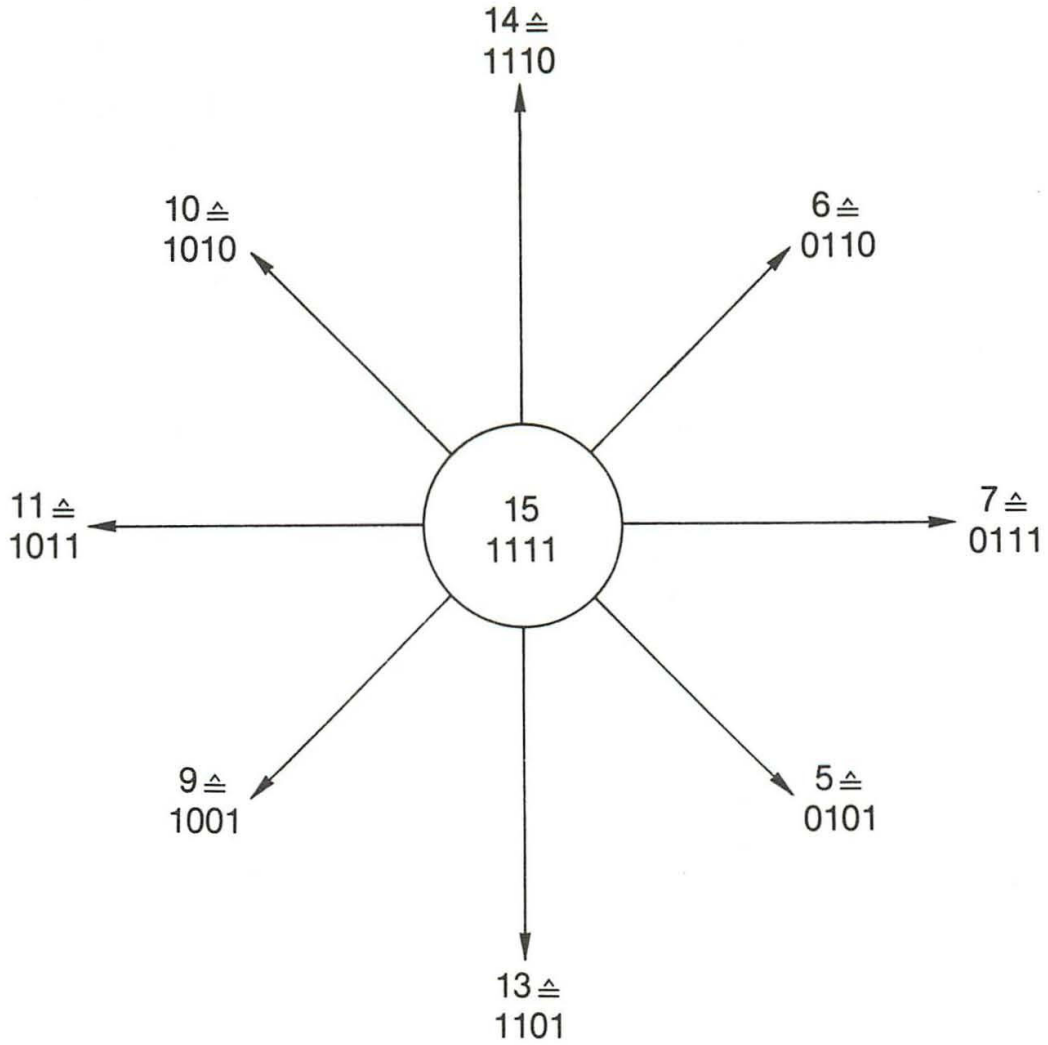
Wir beginnen damit, den Zeichensatz zu kopieren.

```
2530 REM
2540 REM -------------- ZEICHENSATZ KOPIEREN -------------
2550 FOR SCHLEIFE=0 TO 511
2560 POKE ZSATZ+SCLEIFE,PEEK(57344+SCHLEIFE)
```

2570 NEXT SCHLEIFE

Da wir uns in Grafikmode 1 befinden, benötigen wir nur 512 Zeichen aus dem Zeichensatz. Wer möchte, kann an dieser Stelle das Maschinenprogramm aus dem Programm ZNEU abschreiben.

Nun müssen wir einige Zeichen umdefinieren. Dazu eine Vorüberlegung: jedes Zeichen bestimmt die Richtung in die gegangen werden kann. Befinden wir uns z.B. auf den oberen Ende einer Leiter, so kann sowohl nach links, nach rechts, sowie nach unten gegangen werden. Wenn wir dies geschickt ausnützen, so gestaltet sich die Abfrage, wann in welche Richtung gegangen werden kann sehr einfach. Dazu betrachten wir die Werte, die wir auf die Joystick-Abfrage erhalten. Zerlegt man diese Werte in Dualzahlen, so sieht man, daß jeweils eine 0 für eine Richtung verantwortlich ist.



Und folgende Zeichen haben wir neu definiert.

Bewegung möglich in Richtung	Wert	Wert+2	Zeichen (alt)
rechts	7	9	)
links	11	13	-
links,rechts	3	5	%
links,rechts,oben	2	4	$
links,rechts,unten	1	3	#

Damit gestaltet sich die Abfrage, ob das Gehen in diese Richtung möglich ist, sehr einfach. Man muß nur eine logische ODER-Verknüpfung zwischen dem internen Code des Zeichens, auf dem das Männchen steht, und dem Wert, den wir mit Stick(0) erhalten, machen. Diese ODER-Verknüpfung wird in dem Maschinenprogramm in dem String TEST$ realisiert. Befinden wir uns am unteren Ende einer Leiter, so lautet das entsprechende Zeichen '"'. Da wir dieses Zeichen in einem PRINT-Kommando nicht ausgeben können, erhöhen wir die Werte unserer Zeichen um zwei - dadurch ist das Programm etwas leichter zu lesen. Für die restlichen Zeichen, die Häuser, Bäume, usw. verwenden wir die Zeichen mit dem internen Code von 3 bis 14. Unser Programmteil lautet damit:

```
2580 REM -----------------NEUE ZEICHEN -----------------
2590 RESTORE 2650
2600 FOR SCHLEIFE=3 TO 14
2610 FOR ZEILE=0 TO 7
2620 READ BYTE:POKE ZSATZ+8*SCHLEIFE+ZEILE,BYTE
2630 NEXT ZEILE
2640 NEXT SCHLEIFE
2650 DATA 0,0,0,0,0,195,195,255
2660 DATA 195,195,195,255,195,195,195,255
2670 DATA 0,0,0,0,0,0,0,255
2680 DATA 221,217,249,249,249,249,249,255
2690 DATA 255,255,255,217,217,217,217,217
2700 DATA 48,60,60,120,124,252,62,126
2710 DATA 128,128,128,128,128,128,128,255
2720 DATA 24,24,24,24,24,56,60,255
2730 DATA 126,127,55,126,94,124,60,28
2740 DATA 24,24,60,60,126,126,255,255
2750 DATA 1,1,1,1,1,1,1,255
2760 DATA 12,28,28,24,56,60,116,124
```

Nun schalten wir den neuen Zeichensatz ein:

```
2770 POKE 756,ZSATZ/256
```

Bevor wir das Bild zeichnen wählen wir die Farben:

```
2780 REM --------------- BILD ZEICHNEN ------------------
2790 SETCOLOR 0,2,6
2800 SETCOLOR 1,11,10
2810 SETCOLOR 2,0,0
2820 SETCOLOR 3,0,8
```

Jetzt können wir unser Bergwerk zeichnen, wobei wir folgende Darstellung gewählt haben.

```
2830 PRINT #6;"    ó   £ó£ £ , ó £ ";:REM ó = CTRL-N
2840 PRINT #6;"  ý ä   äää ä ý ä ä ";:REM £ = CTRL-H
2850 PRINT #6;")éò%*%%é****%*%ò%*é*-";:REM ý = INV CTRL-G
2860 PRINT #6;" ç      ç          ç  ";:REM ä = CTRL-K
2870 PRINT #6;")ç%%é%%ç%%é%%%%%éç%-";:REM é = INV CTRL-C
2880 PRINT #6;"    ç     ç     ç   ";:REM ç = INV CTRL-D
2890 PRINT #6;")%é%ç%-)%%ç%é%-)ç%%-";:REM ò = INF CTRL-F
2900 PRINT #6;"  ç          ç"
2910 PRINT #6;")éç%%%-)é%%%ç%%%é%%-";
2920 PRINT #6;" ç       ç        ç"
2930 PRINT #6;")ç%é-)%%ç%%%%%-)çé%-";
2940 PRINT #6;"   ç              ç"
2950 PRINT #6;")é%ç-)é%-)%é%%é-)çé-";
2960 PRINT #6;" ç    ç    ç  ç   ç"
2970 PRINT #6;")ç%-)%ç%%%%ç-)ç%%%ç-";
```

Da wir immer das Zeichen abfragen, auf dem das Männchen gerade steht und auf Grund dessen wir das Männchen in vier mögliche Rcihtungen bewegen können, ist der Aufbau des Bildes frei wählbar. Wer möchte kann also an dieser Stelle, um den Spielreiz zu erhöhen, eine Routine einsetzen, die die Leitern und Barrieren zufällig setzt. Jetzt schreiben wir noch in die GRAPHICS-0-Zeile unsere Angaben:

```
2980 PRINT CHR$(125);"SCORE        ZBON        ZEIT";
```

Dabei sollten die drei Wörter invers geschrieben werden. Zwischen den Wörtern befinden sich jeweils 8 Leerzeichen.

Damit können wir das Unterprogramm beenden:

```
2990 RETURN
```

An dieser Stelle kann man im Hauptprogramm eine Endlos-Schleife einsetzen, das Spiel starten, und den Bildschirm nach belieben ändern.

3

Player Missiles

3. Player-Missile

Nachdem wir alle Grafikmodes besprochen haben, und den Video-Prozessor etwas kennengelernt haben, wollen wir uns mit der zweiten Möglichkeit beschäftigen, Objekte auf den Bildschirm zu bringen, die sogenannten Player-Missile (Spieler-Rakete). Der Name deutet schon darauf hin, daß man damit sehr gut gewisse Objekte auf den Bildschirm bewegen kann.

3.1 Definition

Mit unseren bisherigen Mitteln, könnten wir Bewegung eines Männchens nur dadurch herstellen, daß wir ein geeignetes Bit-Muster durch das Bildschirm-RAM schieben. Diese Methode hat jedoch viele Nachteile: Erstens wäre die Darstellung eines solchen Männchens nur in hochauflösender Grafik möglich und zweites wäre eine schnelle Bewegung dieses Männchens in Basic nicht realisierbar, da man hierzu die Geschwindigkeit eines Maschinenprogramms benötigt. Ein weiteres Problem ist die Überdeckung mit dem Hintergrund, denn bei der Verschiebung unseres Männchens müßte der Hintergrund immer wieder aufgefrischt werden. Außerdem ist es schwierig Kollisionen (z.B. Rakete trifft Raumschiff) abzufragen.

Eine Lösung dafür wird durch die Player-Missile geboten. Hier hat man die Möglichkeit bis zu 8 Objekte (4 Player und 4 Missile) auf den Bildschirm zu bringen, die vom Bildschirm RAM völlig unabhängig sind. D.h. wird ein Player bewegt, so muß der Hintergrund nicht 'aufgefrischt' werden, da sowohl alle Player und Missile und auch die normale Grafik getrennt auf den Bildschirm gebracht werden.

Zum Zweiten können mit Hilfe der Player-Missile Überdekkungen leicht abgefragt werden, d.h. es gibt Speicheradressen, die angeben, welche Objekte sich auf dem Bildschirm überdecken. Dabei gibt es auch ein Register, das Überdeckungen mit der Hintergrundgrafik angibt, wobei zwischen den einzelnen Farben unterschieden wird. Zeichnen wir z.B. im Grafik-Mode 5 vier verschiedenfarbige Streifen auf den Bildschirm, und lassen einen Player darüber wandern, so ist das entsprechende Kollisionsregister solange 0, bis der Player auf einen Streifen trifft. Im Kolli-

sionsregister kann man dann ablesen, welche Streifen der Player gerade überdeckt (dies können ja auch mehrere sein).

Zum Dritten kann man bei den Player-Missile wählen, welches Object auf den Bildschirm Priorität vor den anderen hat. Man kann also den Player im obigen Beispiel über zwei Streifen hinwegwandern lassen, und hinter den zwei anderen Streifen verschwinden lassen. Dies ist zwar nicht in beliebiger Weise möglich, doch hat man sehr viele Möglichkeiten.

Die Player-Missile haben allerdings den Nachteil, daß jeder Player nur acht Punkte und jede Missile nur 2 Punkte breit sein kann, und einfarbig ist. Dadurch ergibt sich für die Kodierung einer Zeile eines Players ein Speicherbedarf von einem Byte, und für die Missiles von 2 Bit. Da die Kodierung aller Missiles zusammengefasst wird, ist für die Kodierung einer Zeile aller Missiles ebenfalls ein Speicherbedarf von einem Byte nötig. Wie breit ein Objekt auf dem Bildschirm dargestellt wird, hängt natürlich davon ab, wie breit ein Punkt ist. Die normale Breite eines Punktes entspricht der Breite von zwei Grafik-8-Punkten (COLOR CLOCK; entspricht genau der Breite von zwei Grafik-8-Punkten). Nun kann für alle Objekte getrennt bestimmt werden, ob das Objekt in normaler, doppelter oder vierfacher Breite dargestellt werden soll.

Außerdem kann bestimmt werden, wie hoch ein Punkt auf dem Bildschirm dargestellt wird (dies ist jedoch dann für alle Player und Missile gleich). Dafür gibt es die Möglichkeit einen Punkt eine Scan-Line hoch zu wählen (dies bezeichnen wir in diesem Buch mit Ein-Zeilen-Auflösung), oder doppelt so hoch (Zwei-Zeilen-Auflösung). Da sich alle Objekte in der Höhe über den ganzen Bildschirm erstrecken, ergibt sich ein Speicherbedarf pro Player bzw. für alle Missile von mindestens 192 bzw. 192/2=96 Byte. In Wirklichkeit sind die Speicher für alle Player und Missile 256, bzw. 128 Byte groß. Daraus folgt, daß sich alle Objekte über den oberen bzw. unteren Bildschirmrand hinaus erstrecken können. Damit setzt sich der Speicherplatz für Player-Missile aus 4x256+256=1280 bzw. 4*128+128=640 Byte zusammen.

Bevor wir nun beginnen einen Player für unser Spiel zu definieren, geben wir eine Tabelle aller für Player-Missile wichtigen Register an.

SDMCTL (559)

Dieses Register haben wir schon im letzten Kapitel kennengelernt, als wir mit dem Befehl POKE 559,0 den Video-Prozessor ausschalteten, damit er beim Ändern der Display List nicht 'durcheinandergerät'. Zum Einschalten haben wir den Befehl POKE 559,34 kennengelernt. Zu diesem Wert (34) ist nun folgendes zu addieren:

4	für Missile
8	für Player
16	für Ein-Zeilen-Auflösung

Wird die 16 nicht addiert, so liegt immer Zwei-Zeilen-Auflösung vor.

GRACTL (53277 / nur schreiben)

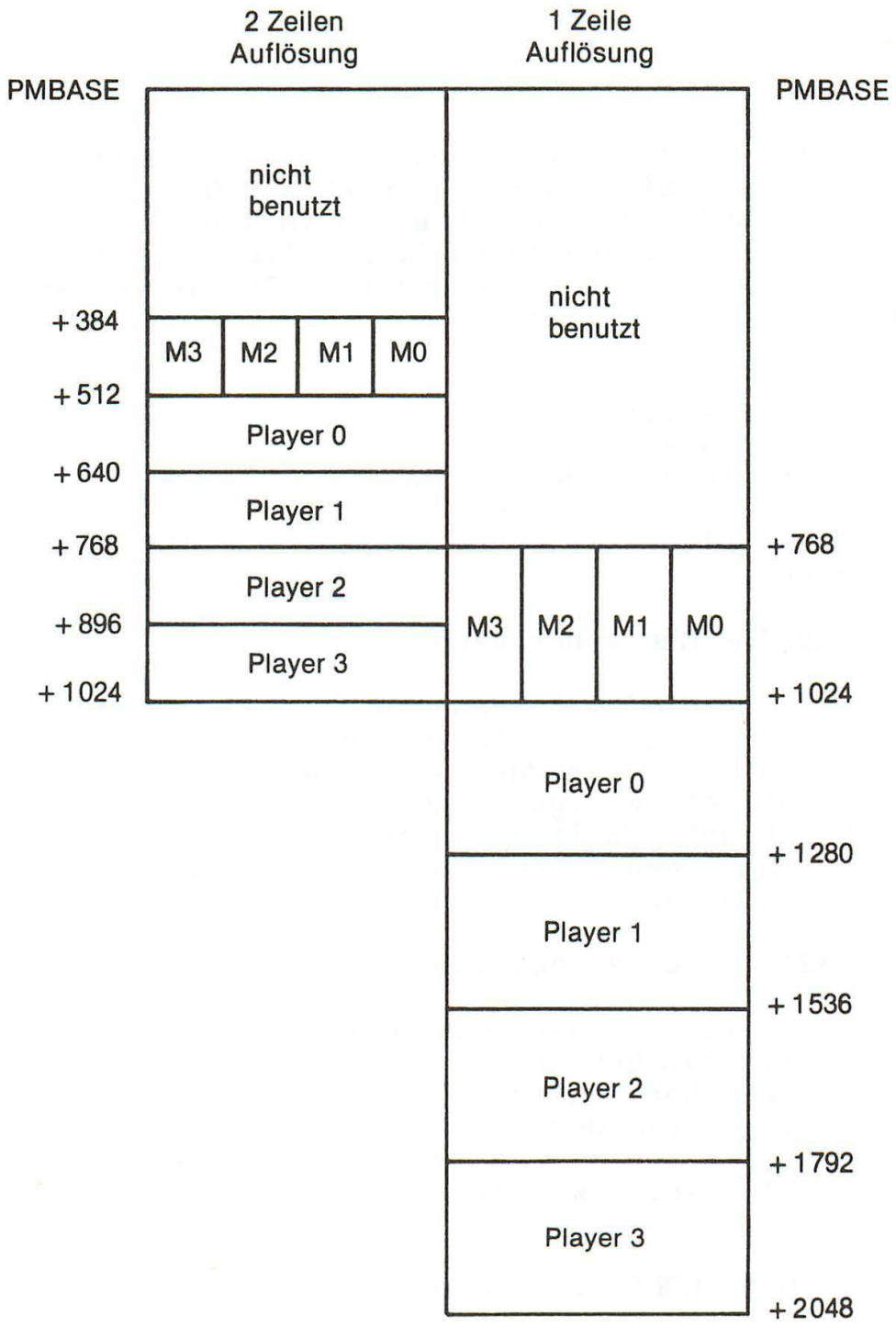
In die Adresse 53277 muß man eine 1 für Missile hineinschreiben, eine 2 für Player, und eine 3 falls sowohl Player als auch Missile dargestellt werden sollen. Dieses Register kann, wie einige später folgende, nur beschrieben werden. Das hängt damit zusammen, daß in diesem Bereich kein RAM im eigentlichen Sinne vorhanden ist, sondern direkt der Video-Prozessor angesprochen wird.

PMBASE (54279 / nur schreiben)

In das Register PMBASE muß man die Start-Adresse des Player-Missile-RAM-Bereichs schreiben (nur MSB). Aus hardwaretechnischen Gründen muß diese Adresse durch 2048 bei Ein-Zeilen-Auflösung, bzw. durch 1024 bei Zwei-Zeilen-Auflösung teilbar sein. Die Player und Missile findet man ab dieser Adresse, wie die Übersicht auf der nächsten Seite zeigt.

COLOR0, COLOR1, COLOR2, COLOR3 (704-707)

In diese Register sind in bekannter Art und Weise die Farben von Player 0 bis 3 und Missile 0 bis 3 zu schreiben. Die einander zugeordneten Player und Missile haben jeweils dieselbe Farbe.


2 Zeilen
Auflösung
1 Zeile
Auflösung
PMBASE
PMBASE
nicht
benutzt
nicht
benutzt
+384
M3
M2
M1
M0
+512
Player 0
+640
Player 1
+768
+768
Player 2
+896
M3
M2
M1
M0
Player 3
+1024
+1024
Player 0
+1280
Player 1
+1536
Player 2
+1792
Player 3
+2048

SIZEP0...SIZEP3 (53256-53259 / nur schreiben)

Hier steht die Breite der Player. Dabei bedeuten 0 oder 2 normale Breite, 1 doppelte Breite und 3 vierfache Breite.

SIZEM (53260 / nur schreiben)

In SIZEM steht die Breite aller Missile. Da die Breite nur Werte zwischen 0 und 3 annehmen kann, sind zur Kodierung der Breite nur 2 Bit erforderlich. In diesem Register stehen daher die Breiten für alle Missile. Dabei ist der Wert für die Breite jeweils mit 1,4,16 oder 64 zu multiplizieren - je nachdem, welches Missile (0-3) man wählt - und die entsprechenden Werte zu addieren.

HPOSP0...HPOSP3 (53248-53251 / nur schreiben)

In diesen Registern steht die horizontale Position der Player. Dabei bedeutet der Wert 0 eine Position die links vom Bildschirm - also nicht sichtbar - ist. Die sichtbaren horizontalen Werte liegen in etwa zwischen 48 und 208.

HPOSM0...HPOSM3 (53252-53255 / nur schreiben)

Dies sind die entsprechenden Positions-Register für die Missile.

VDELAY (53276 / nur schreiben)

Da alle Objekte in der Zwei-Zeilen-Auflösung jeweils nur um zwei Scan-Lines nach oben bzw. nach unten verschoben werden können, kann man in diesem Register für alle Objekte die Verschiebung um eine Scan-Line nach unten erreichen.

Für alle Objekte steht dabei jeweils ein Bit zur Verfügung. Die Zuordnung zeigt folgende Tabelle:

1	Missile 0
2	Missile 1
4	Missile 2
8	Missile 3
16	Player 0
32	Player 1
64	Player 2
128	Player 3

Wird der jeweilige Wert addiert, so verschiebt sich das entsprechende Objekt um eine Scan Line nach unten.

Nun beginnen wir schrittweise einen Player für unser Spiel zu konstruieren.

Zunächst überlege man sich, wieviel Platz man für die Player Missile freihalten muß. Da wir mit Zwei-Zeilen-Auflösung arbeiten wollen, und nur Player definieren, benötigen wir 512 Byte Speicherplatz. Wir hatten für den Zeichensatz insgesamt 1 KByte freigehalten, wovon der Zeichensatz jedoch nur 512 Byte benötigt. Die restlichen 512 Byte können wir somit für unsere Player verwenden. Damit beginnt Player 0 bei Adresse ZSATZ+512.

```
1070 PMSTRT=ZSATZ+512
```

Dann springen wir in ein Unterprogramm, beginnend mit Zeile 3000, in dem wir die Anfangsbedingungen unseres Programmes schreiben, insbesondere auch den Player 0 definieren.

```
1080 GOSUB 3000
3000 REM ************************************************
3010 REM *              ANFANGSBEDINGUNGEN              *
3020 REM ************************************************
```

Nun überlegen wir uns ein Bit-Muster für unseren Player. Das geht relativ gut mit Bleistift und kariertem Papier, da die Player nur einfarbig sind; ansonsten könnte man sich ein Programm ähnlich ZNEU erstellen.

Als nächstes müssen wir dieses Bit-Muster an geeigneter Stelle ins RAM setzen. Da der Player in der Höhe über den ganzen Bildschirm geht, richtet sich also die vertikale Position des Players danach, wohin wir unsere Bit-Muster in den 128 Byte großen Bereich für Player 0 schreiben. Wir lassen die ersten 48 Byte frei, so daß der Player genau auf unsere Erdoberfläche zu stehen kommt.

```
3140 REM ----------------- PLAYER DEFINIEREN ------------
3150 RESTORE 3190
3160 FOR SCHLEIFE=1 TO 7
3170 READ BYTE:POKE PMSTRT+47+SCHLEIFE,BYTE
3180 NEXT SCHLEIFE
3190 DATA 8,28,8,28,28,8,24
```

Nun wählen wir die Farbe:

```
3200 POKE 704,14
```

Wie breit soll der Player werden? Da wir einen Player in normaler Breite wählen, müssen wir dies nicht extra ins Programm schreiben.

Danach schreiben wir die horizontale Position des Player 0 in das entsprechende Register.

```
3210 POKE 53248,168
```

Nun müssen wir uns im Falle der Zwei-Zeilen Auflösung entscheiden, ob und welche Objekte wir um eines Scan Line nach unten verschieben wollen. Da das Männchen noch über dem Erdboden schwebt, setzen wir VDELAY auf den Wert 16.

```
3220 POKE 53276,16
```

Jetzt beginnen wir alle entsprechenden Register zu beschreiben, damit der Player im Bildschirm erscheint. Als erstes schreiben wir die Start-Adresse der Player-Missile in das Register PMBASE.

```
3230 POKE 54279,ZSATZ/256
```

Im folgenden schreiben wir den Wert 2 in das Register GRACTL.

```
3240 POKE 53277,2
```

Als letztes schreiben wir den Wert 46 in das Register SDMCTL, um die Darstellung von Playern mit Zwei-Zeilen-Auflösung zu ermöglichen.

```
3250 POKE 559,46
```

Damit ist die Definition unseres Players beendet, und wir können mit einem RETURN das Unterprogramm verlassen.

```
3260 RETURN
```

Der nächste Player ist der Verletzte. Dies machen wir in einem eigenen Unterroutine ab Zeile 3500.

```
1090 GOSUB 3500
3500 REM ***********************************************
3510 REM *                VERLETZTER                   *
3520 REM ***********************************************
```

Die Definition des Verletzten verläuft analog zur Definition unseres Männchens. Die waagerechte Position auf dem Bildschirm wollen wir dabei zufällig zwischen 148 und 199 festsetzen, und die vertikale Position je nach der Punktezahl.

```
3530 VPOS=8*INT(RND(0)*SCORE/100)
3540 IF VPOS>40 THEN VPOS=40
3550 RESTORE 3610
3560 POKE 53249,48+INT(RND(0)*152)
3570 POKE 705,14
3580 FOR SCHLEIFE=1 TO 3
3590 READ BYTE:POKE PMSTRT+188+VPOS+SCHLEIFE,BYTE
3600 NEXT SCHLEIFE
3610 DATA 90,127,26
3620 RETURN
```

Beim Starten des Programms müßte nun ein stehendes Männchen links vor dem Haus mit Dach und ein liegendes irgendwo im ersten Untergeschoß zu sehen sein.

3.2 Bewegung

Ein waagerechtes Verschieben der Männchen ist leicht zu realisieren, da dabei nur der Wert in den entsprechenden Registern für die horizontale Position geändert werden muß. Dabei ist allerdings zu beachten, daß diese Register nur beschrieben werden können. Man muß also den aktuellen Wert für die horizontale Position jedes Objektes in einer eigenen Variablen speichern. Wir verwenden dazu die Variable X, die allerdings nicht den Wert angibt, der in das Register geschrieben werden muß, sondern der Spalte des Zeichens entspricht, auf dem der Player gerade steht. Dies ist für unsere Zwecke geeigneter, da die möglichen Bewegungen des Männchens von dem Zeichen abhängen, auf dem das Männchen gerade steht.

Eine vertikale Bewegung ist in Basic schwerer zu realisieren, da hierbei das Bit-Muster des Objekts in den für dieses Objekt reservierten Speicherbereich verschoben werden muß. In Basic hat man deshalb keine andere Möglichkeit, als das Bit-Muster aus dem RAM-Bereich zu löschen, und an anderer Stelle neu zu setzen.

Nun können wir die Bewegungs-Routinen in unser Programm einsetzen. Dabei schreiben wir das Programm so, daß die Bewegung mit den Joystick erfolgt (wer keinen Joystick hat kann die Unterroutine aus dem Programm ZNEU verwenden,um das Männchen zu steuern).

Im Spiel geht es also folgendermaßen weiter:

```
1100 REM ------------- BEGINN HAUPTPROGRAMM -------------
1110 ST=STICK(0)
1120 LOCATE X,Y,L
1130 IF L=42 OR L=134 THEN GOSUB 4230:GOTO 1180
1140 GEHT=USR(ADR(TEST$),ST,L-34)
1150 IF GEHT<15 THEN 1170
1160 FOR SCHLEIFE=1 TO 20:NEXT SCHLEIFE:GOTO 1340
1170 GOSUB 4000
1180 IF X<>14 OR Y<>2 OR MANN=0 THEN 1230
1190 SCORE=SCORE+VPOS+50
1200 ZBON=1.8*ZBON
1210 MANN=0
1220 GOSUB 3500
1270 IF X<>2 OR Y<>2 OR ZBON<10 THEN 1340
1280 ZBON=INT(ZBON)
1290 IF ZBON<500 THEN TIME=TIME+ZBON:GOTO 1310
1300 TIME=TIME+500
1310 SCORE=SCORE+TIME
1320 ZBON=1
1330 POKE 657,20:PRINT "      ";
1340 POKE 657,8:PRINT SCORE;
1350 POKE 657,20:PRINT INT(ZBON);
1360 TIME=TIME-1
1370 POKE 657,32:PRINT TIME;" ";
1380 IF TIME=0 THEN 7000
1390 GOTO 1100
3030 DIM TEST$(16)
3040 RESTORE 3080
3050 FOR SCHLEIFE=1 TO 16
3060 READ BYTE:TEST$(SCHLEIFE)=CHR$(BYTE)
3070 NEXT SCHLEIFE
3080 DATA 104,104,133,213,104,133,203,104,104
3090 DATA 5,203,41,15,133,212,96
3100 SCORE=0
3110 ZBON=1
3120 TIME=500
3130 X=15:Y=2
4000 REM ***********************************************
4010 REM *                BEWEGUNGSROUTINEN                *
4020 REM ***********************************************
4030 IF ST<>13 THEN 4130
4040 FOR SCHLEIFE=1 TO 7
4050 POKE PMSTRT+39+4*Y+SCHLEIFE,0
4060 NEXT SCHLEIFE
4070 Y=Y+2
4080 RESTORE 3190
4090 FOR SCHLEIFE=1 TO 7
4100 READ BYTE:POKE PMSTRT+39+4*Y+SCHLEIFE,BYTE
4110 NEXT SCHLEIFE
4120 RETURN
4130 IF ST<>14 THEN 4230
```

```
4140 FOR SCHLEIFE=1 TO 7
4150 POKE PMSTRT+39+4*Y+SCHLEIFE,0
4160 NEXT SCHLEIFE
4170 Y=Y-2
4180 RESTORE 3190
4190 FOR SCHLEIFE=1 TO 7
4200 READ BYTE:POKE PMSTRT+39+4*Y+SCHLEIFE,BYTE
4210 NEXT SCHLEIFE
4220 RETURN
4230 IF ST<>7 THEN 4300
4240 XX=48+8*X
4250 FOR SCHLEIFE=2 TO 8 STEP 2
4260 POKE 53248,XX+SCHLEIFE
4270 NEXT SCHLEIFE
4280 X=X+1
4290 RETURN
4300 IF ST<>11 THEN RETURN
4310 XX=48+8*X
4320 FOR SCHLEIFE=2 TO 8 STEP 2
4330 POKE 53248,XX-SCHLEIFE
4340 NEXT SCHLEIFE
4350 X=X-1
4360 RETURN
```

3.3 Prioritäten

Wie wir am Anfang von Kapitel 3 bereits erwähnten, ist es möglich, zwischen den einzelnen Playern und Missiles und den Hintergrundfarben sozusagen eine 'Hackordnung' einzuführen. D.h. man kann angeben, welches Objekt, bzw. welche Hintergrundfarbe bei Kollision mit einem anderen Objekt Priorität besitzt und damit auf den Bildschirm dargestellt wird. Dazu dient das Register GPRIOR (Adresse 623) über das schon in Zusammenhang mit den GTIA-Modes behandelt wurde. Für die GTIA-Modes sind die höchsten beiden Bit verantwortlich. Die niedrigsten 4 Bit (dezimal 1 bis dezimal 15) dienen zur Festlegung der Priorität.

Dabei gilt:

	Bit D3=1 (dez.8)	Bit D2=1 (dez.4)
Höchste	PF0,PF1	
	P0,P1,P2,P3	PF0,PF1,PF2,PF3
Priorität	PF2,PF3	P0,P1,P2,P3
Niedrigste	Hintergrund	Hintergrund

	Bit D1=1 (dez.2)	Bit DO=1 (dez.1)
Höchste	PO,P1	
	PFO,PF1,PF2,PF3	PO,P1,P2,P3
Priorität	P2,P3	PFO,PF1,PF2,PF3
Niedrigste	Hintergrund	Hintergrund

P steht dabei für Player und PF für Playfield (Hintergrundfarbe). Für die Missiles gilt dasselbe, wie für die entsprechenden Player. Überdecken sich zwei Objekte, die innerhalb einer Zeile stehen, wird das jeweils links stehende im Vordergrund gezeichnet.

Das bedeutet, daß von diesen vier Werten nur einer gewählt werden darf. Wählt man mehrere dieser Werte, so werden die Objekte schwarz in dem Bereich, in dem sie sich überschneiden.

Um nun zu erreichen, daß das Männchen hinter den Bäumen geht, müssen wir also schreiben

```
3205 POKE 623,8
```

Die restlichen beiden Bit haben folgende Funktionen:

Ist das Bit 5 (dezimal 32) 1 gesetzt, so ergibt sich in dem Bereich, wo sich Player 0 mit Player 1 und wo sich Player 2 mit Player 3 überschneidet eine logische ODER-Verknüpfung der Farbwerte im Bereich der Überdeckung. Das bedeutet, daß man mit zwei Playern ein dreifarbiges Männchen zeichnen kann.

Das letzte Bit (dezimal 16) ermöglicht es alle 4 Missile zu einem fünften Player zusammenzufassen. Dabei werden aber die horizontalen Werte für die Position getrennt behandelt, so daß man alle auf einmal entsprechend verändern muß. Ist dieses Bit gesetzt so wird den 4 Missiles (dem 5. Player) die Farbe von Farbregister 712 zugeordnet.

3.4 Kollision

Auch die Kollision wurde schon erwähnt. Es gibt Register, die Kollisionen zwischen den einzelnen Objekten, und auch zwischen Objekten und den Hintergrundfarben anzeigen. Dabei wird zunächst einmal zwischen vier Arten von Kollisio-

nen unterschieden:

- Missile / Hintergrund
- Player / Hintergrund
- Missile / Player
- Player / Player

Da es von jedem Objekt, auch vom Hintergrund, vier Stück gibt, hat man für jede Art der Überdeckung vier Register. Von jedem Register werden dabei die niedrigsten vier Bit benötigt, um anzuzeigen, mit welchem Objekt bzw. mit welcher Hintergrundfarbe die Überdeckung stattfindet. Das niedrigste Bit (Bit 0,dezimal 1) signalisiert dabei eine Überdeckung mit dem Objekt #0. Bit 3 signalisiert eine Überdeckung mit dem Objekt #3. Für die Register gilt folgende Zuordnung:

Register	**Art der Überdeckung**
53248	Missile 0 / Hintergrund
53249	Missile 1 / Hintergrund
53250	Missile 2 / Hintergrund
53251	Missile 3 / Hintergrund
53252	Player 0 / Hintergrund
53253	Player 1 / Hintergrund
53254	Player 2 / Hintergrund
53255	Player 3 / Hintergrund
53256	Missile 0 / Player
53257	Missile 1 / Player
53258	Missile 2 / Player
53259	Missile 3 / Player
53260	Player 0 / Player
53261	Player 1 / Player
53262	Player 2 / Player
52263	Player 3 / Player

Um nun in unserem Spiel festzustellen, ob sich das Männchen mit dem Verletzten überdeckt, müssen wir das Register 53260 abfragen. Da jedes Register 0 ist, falls keine Überdeckung vorhanden ist, müssen wir einfach einen Test auf 0 machen, und folgende Zeile ins Programm einfügen:

```
1230 IF PEEK(53260)=0 THEN 1270
```

Nun muß man das Kollisionsregister löschen. Da die Nummern der Kollisionsregister dieselben sind, wie für die horizontalen Positionen der Objekte und deren Größe, kann man in die Kollisionsregister nicht einfach eine 0 hineinschreiben.

Dazu existiert das Register HITCLR (Adresse 53278). Sobald

dieses Register mit POKE angesprochen wird (d.h. POKE 53278, beliebiger Wert), werden alle Kollisionsregister gelöscht. Dabei ist zu beachten, daß es bis zu einer sechzigstel Sekunde dauern kann, bis der Computer Kollisionen 'bemerkt'. Deshalb sollte die Abfrage auf Kollision möglichst spät hinter der Bewegung der Objekte stattfinden, und das Löschen des Kollisionregister unmittelbar nach dieser Abfrage. In unserem Spiel, müssen wir im Falle der Überdeckung das Kollionsregister löschen, die Variable MANN einsetzen und den Verletzten löschen.

```
1240 POKE 53278,1
1250 MANN=1
1260 GOSUB 4500
```

Als letztes benötigen wir noch eine Routine, um einen Neustart des Spieles zu erreichen.

```
7000 REM ***********************************************
7010 REM *                    NEUES SPIEL              *
7020 REM ***********************************************
7030 GOSUB 4500
7040 FOR SCHLEIFE=1 TO 7
7050 POKE PMSTRT+39+4*Y+SCHLEIFE,0
7060 NEXT SCHLEIFE
7070 IF STRIG(0)=1 THEN 7070
7080 GOSUB 3100
7090 GOSUB 2980
7100 GOTO 1090
```

Nun kann man das Spiel starten. In Kapitel 4 werden wir dann noch eine Möglichkeit aufzeigen, wie die vertikale Bewegung verbessert werden kann.

4

BASIC-Besonderheiten

4. Basic-Besonderheiten

Im letzten Kapitel wollen wir einiges der Besonderheiten des Atari-Basic aufzeigen, auf die auch schon teilweise hingewiesen wurde. Dieses Kapitel ist also als Zusammenstellung einiger nützlicher Tricks und Tips zu verstehen, außerdem werden ein paar Programmierhilfen besprochen und als Programm-Listings angegeben.

4.1 Maschinenroutinen

Es gibt drei vernüftige Möglichkeiten Maschinenroutinen in ein Basic-Programm einzubauen:

1. Man schreibt das Maschinenprogramm in DATA-Zeilen, liest diese Daten während des Programmablaufes in einen String und ruft das Maschinenprogramm mit der USER-Funktion auf. Diese Methode hat den Vorteil, daß das Maschinenprogramm innerhalb des Basic-Speicherbereichs steht, und man sich nicht um einen sicheren Platz kümmern muß. Der Nachteil besteht darin, daß man im Maschinenprogramm keine absoluten Adressen verwenden kann, da man beim Erstellen des Programms noch nicht weiß, wo die Maschinenroutine im Basic-Speicherbereich steht. Diese Methode bietet sich also vor allem für kürzere Maschinenprogramme an.

   Zu bemerken ist noch, daß man sich und dem Rechner viel Arbeit ersparen kann, wenn man das Maschinenprogramm gleich in einen String schreibt. Da die Umrechnung der Dezimalzahlen in die entsprechenden Zeichen für den String relativ mühsam ist, verwendet man am besten folgende Methode, die wir am Beispiel des Zeichensatzkopierprogramms im Programm ZNEU darstellen wollen:

   Als erstes schreibt man einen END-Befehl hinter den Programmteil, wo die Daten für die Maschinenroutine in den String geschrieben werden. Im Programm ZNEU ist dies z.B. die Zeile 1555. Sodann startet man das Programm und läßt sich, nachdem die Ready-Meldung ausgegeben wurde, den entsprechenden String auf den Bildschirm bringen. Da dieser String nun unser gesamtes Maschinenprogramm darstellt, können wir die gesamte Lese-Routine und die Datazeilen

für das Maschinenprogramm durch eine entsprechende Zeile ersetzen. Im Programm ZNEU kann man also die Zeilen 1490 bis 1550 jetzt löschen und stattdessen eine Zeile einfügen, in der steht: COPY$= und dann der String den man sich vorher ausgeben ließ. Diese Methode versagt zunächst, wenn der String ein '"' enthält, oder länger als ca. 100 Zeichen ist. In diesen Fällen muß man den String aufteilen, und alle '"' durch CHR$(32) ersetzen.

2. Wie wir schon gesehen haben, ist der Speicherbereich von 1536 bis 1791 für den Benutzer frei verfügbar. Es bietet sich also an, Maschinenprogramme bis zu einer Länge von 256 Byte an dieser Stelle zu speichern. Da das Maschinenprogramm nun an einer festdefinierten Stelle im RAM liegt, kann das Programm auch absolute Adressen verwenden.

3. Wie wir schon bei den Zeichensätzen und den Player-Missile gesehen haben, kann man sich zwischen den Bildschirmspeicher und dem Basic einen sicheren Speicherplatz schaffen. An dieser Stelle kann man natürlich auch Maschinenprogramme fast beliebiger Länge legen. Wie schon gesehen, ist dabei allerdings zu beachten, daß der Wert in RAMTOP (Adresse 106) nur um Vielfache von 4 und bei Verwendung von Grafik-Modes, die mindestens 4K Bildschirmspeicher benötigen, nur um Vielfache von 16, geändert werden darf. Außerdem ist immer zu beachten, daß hinter dem Bildschirmspeicher bis zu 800 Byte gelöscht, bzw. zerstört werden können.

Wie bekommt man nun ein Maschinenprogramm, das mit Hilfe eines Assemblers auf Diskette erstellt wurde, in DATA-Zeilen? Die Lösung heißt: Adresse 842. Um die Funktion dieser Adresse zu verstehen, müssen wir ein Versuch machen, aber vorher alle wichtigen Daten im Computer **speichern.** Probieren Sie nun folgendes:

Bildschirm löschen und in die vorletzte oder drittletzte Zeile SETCOLOR 4,0,0 ohne RETURN schreiben. Dann mit dem Cursor hinaufgehen und in die erste Zeile den Befehl POKE 842,13 schreiben und RETURN drücken. Was passiert? Der Cursor läuft selbstständig nach unten, und der Bildschirmhintergrund wird schwarz, sobald der Cursor über dem SETCOLOR-Befehl hinwegläuft. Sodann läuft der Cursor weiter nach unten und der Bildschirm wird nach oben gerollt. Diesen Prozess kann man mit der BREAK-Taste nicht anhalten, und teiweise hilft nicht einmal mehr die RESET-Taste.

Folgendes ist passiert: Steht in Speicheradresse 842 der

Wert 13, so simuliert der Rechner fortlaufendes Drücken der RETURN-Taste. Macht man nun dasselbe nocheinmal, schreibt aber statt SETCOLOR 4,0,0 den Befehl POKE 842,12, so hält der Cursor, nachdem er über diese Zeile gelaufen ist wieder an.

Diesen 'Selbst-RETURN-Modus' kann man nun benützen, um per Programm DATA-Zeilen zu erzeugen. Ein solches Hilfsprogramm stellen wir hier vor:

```
1000 REM ***************************************************
1010 REM *              DATAZEILEN GENERATOR               *
1020 REM ***************************************************
1030 PRINT "↖↓DATA-ZEILEN GENERATOR"
1040 PRINT :PRINT :PRINT
1050 PRINT "  STARTADRESSE ";:INPUT START
1060 PRINT "  ENDADRESSE   ";:INPUT EN
1070 PRINT "↓  ERSTE ZEILE  ";:INPUT ZEILE
1080 PRINT "  ZEILENABSTAND";:INPUT ABSTAND
1090 PRINT "↖↓↓"
1100 BILD=10
1110 SCHLEIFE=0
1120 PRINT ZEILE;" DATA ";
1130 PRINT PEEK(START);",";
1140 SCHLEIFE=SCHLEIFE+1:START=START+1
1150 IF START>EN THEN 1170
1160 IF SCHLEIFE<6 THEN 1130
1170 PRINT "◀"
1180 PRINT "CONT";
1190 PRINT "↑↑↑";
1200 POKE 842,13
1210 STOP
1220 POKE 842,12
1230 IF START>EN THEN 1280
1240 ZEILE=ZEILE+ABSTAND
1250 BILD=BILD-1
1260 IF BILD=0 THEN 1090
1270 GOTO 1110
1280 ? "↖↓↓"
1290 POKE 842,13
1300 FOR I=0 TO 30:PRINT 1000+10*I:PRINT "CONT↑↑↑";:STOP
1310 NEXT I
1320 PRINT "↖↓↓1310":? "1320":? "1330"
1330 PRINT "POKE842,12↑↑↑↑↑";:STOP
```

Das Programm funktioniert folgendermaßen: Wir schreiben zuerst die Zeilennummer, das Wort DATA und dann 6 Daten auf den Bildschirm (Zeilen 1120 bis 1160). In Zeile 1170 löschen wir dann das letzte Komma. Darunter schreiben wir das Wort CONT (Zeile 1180), damit das Programm, nachdem die DATA-Zeile erzeugt ist, wieder weiterläuft. Bevor man das Programm nun mit STOP anhält, muß man mit dem Cursor über die geschriebene DATA-Zeile gehen und die 13 in die Adresse 842 schreiben, um den Selbst-RETURN-Modus einzuschalten (Zeilen 1190 bis 1210).

Wird das Programm nun angehalten, so erscheint in der Zeile, in der der Cursor steht, die Meldung STOPPED AT LINE (Zeilennummer). Der Cursor geht dann um eine Zeile nach unten, also in die Zeile, in der die Programmzeile mit der DATA-Anweisung steht. Diese Zeile wird nun in das Basic-Programm eingefügt, der Cursor geht wiederum eine Zeile nach unten, der Rechner empfängt den CONT-Befehl und das Programm wird nach dem STOP fortgesetzt. Zu beachten ist dabei, daß der CONT-Befehl das Programm immer in der nächsten Zeile fortsetzt, d.h. nach den STOP-Befehl darf die Programmzeile keinen weiteren Befehl mehr enthalten.

Auf diese Art und Weise werden alle nötigen DATA-Zeilen erzeugt, und dann das Programm gelöscht, indem alle Zeilennummer nacheinander auf den Bildschirm gebracht, und im 'Selbst-RETURN-Modus' einzeln gelöscht werden (Zeilen 1280 bis 1330).

Wir schreiben deshalb nur 6 Daten in eine DATA-Zeile, damit wir sicher sein können, daß eine Programmzeile nicht länger als 38 Zeichen wird. Wäre dies der Fall, so müßte man nach der Ausgabe des CONT-Befehls mit dem Cursor um eine Zeile weiter nach oben gehen, damit die STOP-Meldung an der richtigen Stelle steht. Da die Daten aber zwischen einem und 3 Zeichen lang sein können, kann man in diesen Fällen nicht mehr genau entscheiden, ob die DATA-Zeile länger oder kürzer als 39 Zeichen ist.

4.2 XIO - Befehl

Daß man mit dem XIO-Befehl viel mehr machen kann, als Flächen ausmalen, wurde schon erwähnt. Das allgemeine Format für den XIO-Befehl lautet:

XIO Kommando, Numberzeichen, VAR, VAR1, VAR2,Filename

Dabei bedeuten im einzelnen:

Kommando	Eines der unten angebenen Kommandos
VAR	Nummer des geöffneten Kanals (wie z.B. im OPEN-Befehl)
VAR1,VAR2	Zwei Hilfswerte; normalerweise Null
Filename	Einzugebener Filename

Liste der Kommandos:

Kommando	Operation	Beschreibung
3	OPEN	wie Basic
5	GET RECORD	wie Basic INPUT
7	GET CHARACTER	wie Basic GET
9	PUT RECORD	wie Basic PRINT
11	PUT CHARACTER	wie Basic PUT
12	CLOSE	wie Basic CLOSE
13	STATUS	wie Basic STATUS
17	DRAW LINE	wie Basic DRAWTO
18	FILL	s. Kap. 2.2
32	RENAME	wie DOS RENAME
33	DELETE	wie DOS DELETE
35	LOCK FILE	wie DOS LOCK FILE
36	UNLOCK FILE	wie DOS UNLOCK FILE
37	POINT	wie Basic POINT
38	NOTE	wie Basic NOTE
254	FORMAT	wie DOS FORMAT DISK

Mit den XIO-Befehl XIO 254,#1,0,0,"D:" kann man also eine Diskette formatieren, falls man ein langes Programm geschrieben hat, aber keinen Platz zum Speichern mehr findet.

4.3 Stringinitialisierung

Sehr häufig hat man das Problem, einen längeren String mit ein und demselben Zeichen aufzufüllen. Eine Möglichkeit ist, den gesamten String in das Programm zu schreiben, was natürlich eine sehr lästige und speicherplatzverschwendende Methode ist. Die andere Möglichkeit, den String mit einer FOR...NEXT-Schleife zu füllen, ist sehr langsam.

Die beste Methode ist deshalb die in dem Programm ZNEU verwendete. Die Initialisierung geht dabei nach folgendem Schema:

```
1. STRING$(1)='Zeichen'
2. STRING$('Länge')='Zeichen'
3. STRING$(2)=STRING$
```

Das Prinzip dieser Methode ist sehr einfach zu verstehen: Bei Befehlen wie A$=B$ wird jedes Zeichen von B$ nach A$ kopiert. Das Kopieren beginnt dabei bei dem ersten Zeichen von B$. Schreibt man nun A$(2)=A$ und das erste Zeichen von A$ ist z.B. 'X', so wird bei dem ersten Kopierschritt das zweite Zeichen von A$ das 'X'. In dritten Schritt wird auf die selbe Art und Weise das dritte Zeichen von A$ zum 'X'. Dieses Kopieren einzelner Zeichen geht solange weiter, bis das Ende von A$ erreicht ist. Daher der Befehl A$('Länge')='Zeichen', womit dem Basic gesagt wird, wie lang der String A$ sein soll (die Aktuelle Länge eines Strings wird nicht durch die Dimensionierung bestimmt, sondern durch das höchste Zeichen im String).

4.4 Zeilen löschen

Mit derselben Methode, die wir oben schon verwendet haben, um DATA-Zeilen zu erzeugen, kann man selbstverständlich auch Zeilen löschen. Dieses Verfahren hat jedoch den Nachteil, wenn der Abstand der Zeilennummern nicht ein konstanter Wert ist, daß man theoretisch jede Zeilennummer auf den Bildschirm bringen muß, um diese Zeile eventuell zu löschen. Will man auf diese Art und Weise einen Bereich von Zeile 1000 bis Zeile 2000 löschen, so ist dies ein langwieriger Prozess.

Eine bedeutend elegantere Methode ist es, im Basic-ROM vorhandene Routinen zu benützen, um ganze Zeilenblöcke zu löschen. Wir geben hier ein Hilfsprogramm an, das man bei Bedarf immer mit ENTER zu einem vorhandenen Programm dazu laden kann, um ganze Programmblöcke zu löschen. Wie oben erwähnt, benützen wir dazu Maschinenroutinen, die im Basic stehen. Aus Platzgründen kann deshalb die Funktionsweise des Programms nicht erläutert werden.

Hier das Listing:

```
32600 RESTORE 32604
32601 FOR XI=0 TO 85
32602 READ XBYTE:POKE 1536+XI,XBYTE
32603 NEXT XI
32604 DATA 104,165,138,133,203,165
32605 DATA 139,133,204,104,133,161
32606 DATA 104,133,160,32,162,169
32607 DATA 104,133,161,104,133,160
32608 DATA 32,162,169,176,14,160
32609 DATA 2,177,138,101,138,133
32610 DATA 138,165,139,105,0,133
```

```
32611 DATA 139,165,138,56,229,190
32612 DATA 168,133,212,165,139,229
32613 DATA 191,133,213,165,203,56
32614 DATA 229,212,133,203,165,204
32615 DATA 229,213,133,204,162,138
32616 DATA 165,213,32,250,168,165
32617 DATA 203,133,138,165,204,133
32618 DATA 139,96
32619 XI=USR(1536,32600,32618)
32620 PRINT "↖↓↓32619":PRINT "32620":PRINT "32621"
32621 PRINT "POKE 842,12↑↑↑↑↑";:POKE 842,13:STOP
```

Das Programm läuft folgendermaßen: Zuerst muß man das Programm 'intialisieren', d.h. nach dem Laden muß ein GOTO 32600 eingegeben werden. Dann wird ein Maschinenprogramm erzeugt, das ab Adresse 1536 steht und ab diesem Zeitpunkt immer genutzt werden kann. Nach Erstellen des Maschinenprogramms löscht sich das gesamte Programm selbst. Will man nun einen Programmblock, beginnend mit Zeilennummer A (Startzeile) bis Zeilennummer B (Endzeile) löschen, so muß man folgendes eingeben:

PRINT USR(1536,A,B)

Existieren die angegebenen Zeilennummern nicht, so wird die erste Zeile nach der Startzeile genommen, und die letzte Zeile vor der Endzeile. Da dieses Programm keinerlei Speicherplatz benötigt, kann man es am Anfang der Programmierarbeit immer laden und intialisieren, so daß es während des ganzen Editierens vorhanden ist. Da das Programm in dem Speicherbereich von 1536 bis 1792 steht, geht die Information auch beim Drücken der RESET-Taste nicht verloren, so daß das Programm genützt werden kann, bis man den Rechner ausschaltet.

So wie das Programm aufgebaut ist, läuft es leider nur auf den XL Rechnern. Wer einen Atari 400 oder 800 besitzt, muß in Zeile 32616 den Wert 250 auf 253 ändern.

4.5 Tabellen und Zeiger

Wir möchten an dieser Stelle etwas näher auf das Atari-Basic eingehen, weil es viele Möglichkeiten gibt das Basic für eigene Zwecke zu nützen.

Offensichtlich gibt es zwei grundverschiedene Möglichkeiten ein Programm zu speichern. Einmal mit dem SAVE- , und

zum zweiten mit dem LIST-Kommando. Die beiden Möglichkeiten unterscheiden sich zunächst darin, daß ein mit dem LIST-Kommando gespeichertes Programm beim Laden (ENTER-Kommando) ein schon vorhandenes Programm nicht löscht. Doch die Unterschiede zwischen diesen beiden Kommandos gehen noch viel weiter. Um zu sehen wie SAVE arbeitet, können wir hinter diesem Kommando als Filename "S:" angegeben. Dazu geben wir folgendes ein:

```
NEW
10   REM  ***********   Demo   **************
20   VARIABLE=1
```

Mit dem Kommando SAVE "S:" sieht man nun, was normalerweise auf Diskette, bzw. Kassette gespeichert wird. Außer der REM=Zeile und dem Wort VARIABLE, erkennt man vom Programm nicht viel. Dies liegt daran, daß das Basic beim Eingeben der Programmzeilen eine Vorübersetzung macht, d.h. alle Basic-Kommandos wie z.B. PRINT, SIN, werden in sogenannte TOKENs übersetzt. Dadurch wird erstens Platz gespart und zweitens das Programm schneller. Man sieht auch auf dem Bildschirm, daß die Variable VARIABLE vor der REM-Zeile auftritt. Um dies genauer zu untersuchen, geben wir folgendes ein:

```
30  VAR=2:PRINT VAR
```

Macht man wieder SAVE"S:", so sieht man die Variable VAR direkt hinter dem Namen VARIABLE auftauchen. Hinter dem Programmteil, der mit der REM-Zeile 10 beginnt, kommen die Variablennamen offensichtlich nicht mehr vor. Dies liegt daran, das beim Vorübersetzen die Variablennamen alle in eine Tabelle eingetragen werden, und im eigentlichen Programm nur noch ein Zeiger auf diese Variable steht. Da die Variable VAR hinter VARIABLE steht, werden die Variablennamen offensichtlich so eingetragen, wie sie beim Programmieren zeitmäßig aufeinander folgen. Da das Basic wissen muß, wann ein Variablennamen aufhört, werden die letzten Zeichen der Namen invers geschrieben. Bisher haben wir nur einfache Variablen (Skalare) benutzt. Um zu sehen wie nun die Namen für dimensionierte Variablen und Zeichenketten abgespeichert werden, geben wir folgendes ein:

```
40 DIM STRING$(1),VEKTOR(2),MATRIX(3,4)
```

Wir machen erneut SAVE"S:". Man erkennt, daß bei den dimensionierten Variablen offensichtlich die Klammer, und bei den Strings das '$'-Zeichen in die Namen mit aufgenommen werden. Allerdings ist aufgrund der Variablennamen VEKTOR(und MATRIX(nicht zu unterscheiden, ob es sich um

eine einfach-, oder um eine zweifach indizierte Variable handelt. Wie das Basic hier unterscheidet, werden wir gleich sehen.

Weiterhin fällt auf, daß Bereich zwischen den Variablennamen und dem eigentlichen Programmbeginn, d.h. dem REMARK mit den vielen Sternen, immer größer wird. Dieser Bereich wird genutzt, um die aktuellen Werte der Variablen zu speichern. Dazu sind für jede Variable 8 Byte reserviert. Die Reihenfolge ist dabei dieselbe wie in der Tabelle der Variablennamen. Die Bedeutung dieser 8 Byte pro Variable zeigt folgende Tabelle.

<table>
<tr><th>Byte / Typ</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th></tr>
<tr><td>Skalar</td><td rowspan="3">Typ</td><td rowspan="3">Num-mer</td><td colspan="6">Wert</td></tr>
<tr><td>Array</td><td colspan="2">Zeiger</td><td colspan="2">1. Dimens.</td><td colspan="2">2. Dimens.</td></tr>
<tr><td>String</td><td colspan="2">Zeiger</td><td colspan="2">Länge</td><td colspan="2">Dimens.</td></tr>
</table>

Die ersten beiden Byte geben also für jede Variable den Typ und die Nummer an. Bei Skalaren, d.h. 'einfachen' Variablen, wie z.B. VAR) geben dann die letzten 6 Byte den Wert an. Das bedeutet, daß jede Variable 6 Byte Speicherplatz verbraucht, egal, ob sie einen Wert zwischen 0 und 255 oder irgendeinen Fließkommawert beinhaltet. Da die 8 Byte sowohl für die Arrays (das sind die einfach und zweifach dimensionierten Variablen) als auch für die Strings nicht ausreicht, steht an dieser Stelle für jede solche Variable ein Zeiger. Dieser 2-Byte-Zeiger deutet

aber nicht direkt auf die Speicheradresse, wo der aktuelle Wert beginnt, sondern er gibt nur den Abstand an, zwischen dem Beginn einer Tabelle, in der alle Array- und String-Werte stehen und dem eigentlichen Beginn des Variablenwertes. Im Falle eines Arrays geben die letzten 4 Byte die bei-en Dimensionierungen an, was bedeutet, daß ein Vektor auch als zweidimensionales Array abgespeichert wird, wobei die zweite Dimension immer auf '1' gesetzt ist. Im Falle eines Strings geben die Byte 5 und 6 die aktuelle Länge an, und die letzten beiden Byte die Dimensionierung.

Wir haben bisher vier große Bereiche kennengelernt:

- die sogenannte Variablennamentabelle
- die Variablenwerttabelle
- den Programmbereich
- ein Bereich in dem alle Strings und Array-Werte gespeichert sind.

Dazu gibt es für jede dieser Tabelle einen Zeiger:

Name	Adresse	Bedeutung
LOMEM	128,129	Beginn Basic
VNTP	130,131	Zeiger auf die Variablennamentabelle
VNTD	132,133	Zeiger auf das Ende der Variablennamentabelle
VVTP	134,135	Zeiger auf die Variablenwerttabelle
STMTAB	136,137	Zeiger auf das 'Programm'
STMCUR	138,139	Zeiger auf aktuelles Statement
STARP	140,141	Zeiger auf den String-, Array-Bereich
RUNST	142,143	Beginn Runtimestack
MEMTOP	144,145	Ende Basic

Den Zeiger LOMEM haben wir schon kennengelernt, als wir uns die Speicherbelegung des Rechners angeschaut haben. Dieser Zeiger deutet auf den niedrigsten, nicht vom Betriebssystem genutzten Speicherplatz. Ab dieser Adresse sind 256 Byte frei, die das Basic intern als Pufferspeicher benutzt. Der zweite Zeiger VNTP zeigt auf den Beginn der uns schon bekannten Variablennamentabelle. Der Zeiger VNTD zeigt auf das Ende dieser Tabelle. Der Zeiger VVTP zeigt auf den Beginn der Variablenwerttabelle. STMTAB ist ein Zeiger auf den Beginn des Programmbereichs, d.h. auf die Adresse der ersten Zeilennummer. Der folgende Zeiger STMCUR, wird vom Basic beim Programmlauf benutzt, und zeigt immer auf den Beginn eines Statements. Der Zeiger STARP zeigt auf den Beginn des Bereiches, wo alle String und Array-Werte abgespeichert sind. RUNSTK zeigt auf den Beginn des Kellersspeichers, in dem das Basic gewisse Werte für Rücksprung-Adressen (GOSUB...RETURN) und Werte

für FOR...NEXT-Schleifen ablegt. Der Zeiger MEMTOP zeigt schließlich auf den ersten freien Speicherplatz, der von Basic nicht benutzt wird. Dieser Platz verschiebt sich während des Programmlaufs, da sich die Größe des Basic-Kellerspeichers während des Programmlaufs ständig verändert.

Wie können wir nun diese Tabellen für unsere Zwecke nützen? Wir wollen damit beginnen, alle im Programm vorhandenen Variablennamen auszugeben. Zuerst das Programm:

```
32600 GRAPHICS 0
32601 XVNTP=PEEK(130)+256*PEEK(131)
32602 XVNTD=PEEK(132)+256*PEEK(133)
32603 FOR XVZGR=VNTP TO VNTD-1
32604 XCH=PEEK(XVZGR)
32605 IF XCH<128 THEN PRINT CHR$(XCH);:GOTO 32607
32606 PRINT CHR$(XCH-128)
32607 NEXT XVZGR
```

In Zeile 32600 löschen wir den Bildschirm. In den Zeilen 32601 und 32602 bestimmen wir die aktuellen Werte für den Zeiger auf die Variablennamentabelle (VNTP) und den Zeiger auf das Ende dieser Tabelle (VNTD). Da die Variable VNTD immer auf das erste Zeichen hinter der Variablennamentabelle zeigt, durchlaufen wir die Schleife von VNTP bis VNTD-1. In Zeile 32604 wird das erste Zeichen gelesen. Solange dieses Zeichen nicht invers ist, wird dieses Zeichen auf den Bildschirm gebracht, und dahinter ein ';' gesetzt, das heißt keine neue Zeile angefangen. Ist die Nummer des Zeichens größer bzw. gleich 128, so wird in Zeile 32606 das entsprechende normale Zeichen ausgegeben und kein ';' mehr gesetzt, so daß der Cursor an den Anfang der nächsten Zeile springt.

Dieses Programm ist nicht sehr hilfreich, da man denselben Zweck eigentlich immer mit dem Kommando SAVE"S:" erreicht. Doch kann es passieren, daß bei dem Kommando SAVE"S:" das Steuerzeichen 'Bildschirm löschen' ausgegeben wird, so daß man keine Zeit hat, mit Control/1 die Ausgabe anzuhalten. In solchen Fällen ist dieses kleine Programm sehr nützlich.

Da alle Variablennamen in einer Tabelle stehen, ist es relativ einfach diese mit POKE zu ändern. Auch dazu geben wir ein kleines Hilfsprogramm an. Wie oben zuerst das Programmlisting:

```
32600 DIM NAME$(128)
32601 PRINT "↑↓ VARIABLENNAME ";
32602 INPUT NAME$
32603 VNTP=PEEK(130)+256*PEEK(131)
32604 VNTD=PEEK(132)+256*PEEK(133)
32605 J=VNTP
32606 IF J=VNTD THEN PRINT "GIBT ES NICHT":END
32607 NAMEADR=J:I=0
32608 I=I+1
32609 CH=PEEK(J)
32610 IF CH>127 THEN CH=CH-128
32611 IF CHR$(CH)<>NAME$(I,I) THEN 32650
32612 IF PEEK(J)<128 THEN J=J+1:GOTO 32608
32613 PRINT "NEUER NAME ";
32614 INPUT NAME$
32615 IF LEN(NAME$)=J-NAMEADR+1 THEN 32617
32616 PRINT "FALSCHE LAENGE":GOTO 32613
32617 FOR I=1 TO J-NAMEADR
32618 POKE NAMEADR+I-1,ASC(NAME$(I))
32619 NEXT I
32620 POKE J,ASC(NAME$(I))+128
32621 END
32650 IF PEEK(J)<128 THEN J=J+1:GOTO 32650
32660 J=J+1:GOTO 32606
```

Dieses Programm zerfällt im wesentlichen in zwei Teile. Der Hauptteil besteht darin, einen eingegebenen Variablennamen in der Tabelle zu suchen (32600 bis 32612 und 32650, 32660). Hierbei ist die Variable J unser Parameter, mit dem wir die Variablennamentabelle durchsuchen. Der Wert für J wird also zunächst auf den Beginn der Variablennamentabelle gesetzt (32605). In Zeile 32606 findet die Abfrage statt, ob das Ende der Variablennamentabelle erreicht ist. In diesem Fall geben wir eine Meldung aus und beenden das Programm. Die Variable NAMEADR ist immer die Anfangsadresse des gerade behandelten Variablennamens. Diese Variable erhält zunächst ebenfalls den Beginn der Variablennamentabelle.

Die Variable E ist ein Index auf den String NAME$. In Zeile 32609 lesen wir ein Zeichen aus der Tabelle und wandeln es nötigenfalls in Zeile 32610 in das entsprechende normale Zeichen um. In Zeile 32611 erfolgt die Abfrage, ob der in der Tabelle gelesene Name bis jetzt noch mit den eingebenen Namen übereinstimmt. Ist dies der Fall, so wird in Zeile 32612 erfragt, ob schon das ganze Wort gelesen ist. Ist dies nicht der Fall, so verzweigt das Programm nach Zeile 32608, wo das nächste Zeichen aus der Tabelle gelesen wird. War das Ende des Namens erreicht, so wird in den Zeilen 32613 bis 32621 der neue Name eingegeben, wobei

darauf geachtet werden muß, daß die Länge des neuen Namens mit der Länge des alten Namens übereinstimmt, da das Basic sonst seine Variablen nicht wieder findet. In Zeile 32620 wird schließlich das letzte Zeichen des Namens eingetragen. Die Zeile 32650 ist eine Schleife, die solange durchlaufen wird, bis das Ende des Variablennamens gefunden ist. Diese Zeile wird angesprungen, um das Ende eines Variablennamens in der Tabelle zu finden, falls dieser in den ersten Zeichen nicht mit den eingegebenen Variablennamen übereinstimmt.

Da in den neuen Variablennamen alle Zeichen verwendet werden können, kann man sein Programm dadurch leichter lesbar machen. Allerdings sollte man darauf achten, daß man keine inversen Zeichen eingibt, da dies für das Basic das Ende des Namens bedeutet. Alle anderen Zeichen sind erlaubt, allerdings ist es nicht angebracht Rechenzeichen oder sonstige Zeichen, die im Basic eine besondere Bedeutung haben, zu verwenden. Z.B. kann man in einem Namen statt dem Bindestrich das Zeichen SHIFT '-' verwenden.

Sehr häufig benötigt man beim Programmieren Routinen, die Speicherbereiche verschieben, wie z.B. den Zeichensatz vom ROM ins RAM zu kopieren, und auch beim senkrechten Verschieben von Player-Missile. Hier bietet Basic eine sehr gute Möglichkeit, dies sehr schnell auszuführen.

Wo gibt es so eine Routine? Dazu überlegen wir uns, was passiert, wenn wir die Zeile A$=B$ eingeben und starten. Um diesen Befehl auszuführen, muß Basic jedes Byte von B$ nach A$ kopieren. Dies ist aber nichts anderes als die von uns gewünschte Verschieberoutine. Um nun z.B. den Zeichensatz von ROM ins RAM zu kopieren, benötigt man zwei Strings, der eine mit der Startadresse 57344 (Beginn des Zeichensatzes in ROM) und der andere an einer beliebigen Stelle in RAM. Wie können wir nun dem Basic glaubhaft machen, daß zwei solche Strings existieren?

Dazu betrachten wir noch einmal die Variablenwerttabelle. Man sieht, daß beim Typ String das dritte und vierte Byte ein Zeiger auf die Adresse ist, an der der String steht (plus Wert in STARP). Wenn wir den Wert in diesen beiden Byte nun ändern, können wir es erreichen, daß die beiden Strings an den von uns gewünschten Adressen liegen. Dazu muß man aber zunächst zwei Strings dimensionieren. Da es wenig vorteilhaft ist, zwei Strings der Länge 1024 zu dimensionieren, dimensionieren wir beide Strings kleinst möglichlich d.h. mit der Länge 1. Dann können wir mit POKE-Befehlen sowohl die Dimensionierung, wie auch die aktuelle Länge, und den Eintrag für den Zeiger ändern, so daß Basic glaubt, die beiden Strings liegen an den von uns

gewünschten Adressen. Ist dies geschehen, so wird mit A$=B$ der gesamte Zeichensatz in kurzestmöglicher Zeit kopiert. Betrachten wir dazu folgendes Programm:

```
1000 POKE 106,PEEK(106)-4:GRAPHICS 0
1010 DIM A$(1),B$(1)
1020 VVTP=PEEK(134)+256*PEEK(135)
1030 STARP=PEEK(140)+256*PEEK(141)
1040 DIFF=PEEK(106)*256-STARP
1050 POKE VVTP+2,DIFF-256*INT(DIFF/256)
1060 POKE VVTP+3,INT(DIFF/256)
1070 POKE VVTP+4,0
1080 POKE VVTP+5,4
1090 POKE VVTP+6,0
1100 POKE VVTP+7,4
1110 DIFF=57344-STARP
1120 POKE VVTP+10,DIFF-256*INT(DIFF/256)
1130 POKE VVTP+11,INT(DIFF/256)
1140 POKE VVTP+12,0
1150 POKE VVTP+13,4
1160 POKE VVTP+14,0
1170 POKE VVTP+15,4
1180 POKE 756,PEEK(106)
1190 A$=B$
```

In Zeile 1000 schaffen wir uns zunächst auf die übliche Art und Weise Platz für den neuen Zeichensatz. Dann folgen die Dimensionierungen der beiden Strings. Wie oben schon erwähnt definieren wir die Strings mit der Länge 1, um möglichst wenig Speicherplatz zu verschenken. In Zeile 2020 und 2030 errechnen wir die Werte für die von uns benutzten beiden Basic-Zeiger. In Zeile 1040 errechnen wir für den ersten String A$ die Differenz zwischen dem Beginn des Zeichensatzes und dem Beginn des Stringbereiches. Dieser Wert ist der, der dann in die Variablenwerttabelle eingetragen werden muß. Dieser Eintrag erfolgt in Zeilen 1050 und 1060. In den Zeilen 1070 bis 1100 werden dann die Länge und die Dimensionierung des Strings auf 1024 gesetzt. Die Zeilen 1110 bis 1170 machen dasselbe für den String B$, wobei die Differenz, die in die Variablenwerttabelle eingetragen werden muß, diesmal durch die Anfangsadresse des Zeichensatzes im ROM (57344) und dem Beginn des Stringbereiches bestimmt wird. In Zeile 1180 schalten wir schließlich auf den neuen Zeichensatz um. In Zeile 1190 wird dann der Zeichensatz kopiert. Um zu sehen wie schnell das Kopieren als solches funktioniert, kann man zunächst einmal die Zeile 1190 weglassen, und das Programm starten. Da der neue Zeichensatz schon eingeschaltet wird,

ist nach Beendigung des Programms auf dem Bildschirm nichts zuerkennen. Gibt man nun 'blind' den Befehl A$=B$ ein, so erkennt man, wie schnell der Zeichensatz auf diese Art und Weise kopiert wird.

In diesem Programm hatten wir den Vorteil, daß die zuerst eingetippte Zeile 1010 die Variablen A$ und B$ erhält, so daß wir sicher sein können, daß die beiden Variablen die ersten Variablen in der Variablenwerttabelle sind. Was macht man nun, wenn man diese Routinen nachträglich in ein Programm einbauen will? Dazu genügt es selbstverständlich nicht, die Dimensionierung der beiden Strings in die erste Zeile zu setzen, da dann die Variablennamen, und die Variablenwerte an den Schluß dieser Tabellen gesetzt würden (die zeitliche Reihenfolge beim Eingeben ist wichtig). Um sich langes Suchen in der Variablennamentabelle zu ersparen, gibt es folgende Möglichkeit:

Man schreibt die Dimensionierungen in die erste Zeile des Programms (auf jeden Fall in die erste der Zeilen, wo Variablennamen auftreten). Sodann schreibt man das Programm mit dem LIST-Befehl auf Diskette oder Kassette. Sodann löscht man den gesamten Speicher mit NEW. Lädt man das Programm dann wieder mit dem ENTER-Befehl, so werden die Zeilen einzeln geladen, vorübersetzt, und alle Einträge nacheinander in den Basic-Programm-Bereich übernommen. Diese Methode funktioniert also genau so, als ob man alle Programmzeilen von Anfang an eintippen würde. Auf diese Art und Weise erreicht man, daß die Variablennamen in der Reihenfolge in die Variablennamentabelle bzw. die Werte in die Variablenwerttabelle eingetragen werden, wie sie im Programm aufeinanderfolgen. Diese Methode die auch Atari zur 'Reorganisation des Basicspeichers' angibt, ist auch nützlich, um Variablen, die nicht mehr benötigt werden, aus den Tabellen zu löschen. Auf diese Art und Weise kann also auch Speicherplatz gewonnen werden.

Eine zweite Möglichkeit Verschieberoutinen zu benutzen, ist die vertikale Bewegung der Player-Missile. Da wir in dem Spiel die vertikale Bewegung momentan so darstellen, daß wir das Bitmuster des Männchens an der alten Stelle aus dem Player-Missile-Bereich löschen, und an der neuen Stelle wieder setzen, ist die Bewegung weder sehr elegant noch schnell. Stellen wir die Bewegung allerdings mit dem String-Trick dar, so werden wir sehen, daß wir zwischen den einzelnen Bewegungsphasen noch Leerschleifen einbauen müssen, damit die Bewegung nicht zu schnell abläuft.

Hier nun die Vorgehungsweise im einzelnen:

Als erstes definieren wir einen String der Länge 1. Dieser

String PM$ wird dann, wie oben, an die Adresse gesetzt, wo alle Player-Missile stehen. Die einzelnen Bewegungsphasen der Player schreiben wir in kleine Strings, die an beliebiger Stelle (d.h. an die Stelle wo das Basic sie hinsetzt), gelegt werden können. Dabei ist zu beachten, daß wir bei einer vertikalen Bewegung die Bitmuster nicht löschen und dann neu setzen wollen, sondern gleich den alten Player überschreiben wollen. Dazu überlege man sich, wie groß ein einzelner Schritt bei einer vertikalen Bewegung maximal sein kann. In unserem Beispiel wird die vertikale Bewegung pro Bewegungsphase 4 Scan-Lines sein, d.h. wir verschieben das Bitmuster um zwei Bit im Player-Missile Bereich. Deshalb werden die Player statt 7 Punkte 11 Punkte hoch, wobei die ersten beiden Byte und die letzten beiden Byte 0 gesetzt werden. Dadurch ist gesichert, daß bei einer vertikalen Bewegung das neue Männchen das alte Männchen im Player-Missele-Speicherbereich überschreibt, so daß bei der Bewegung keine 'Reste' entstehen.

Die Bewegung selbst erreicht man nun, indem man einen String, in dem ein Männchen steht, an die entsprechende Stelle in den String PM$ kopiert. Dies geschieht also mit einem Befehl wie z.B. PM$(Y)=P1$.

Unser Programm ändert sich also wie folgt:

Als erstes müssen wir den String PM$ in das Programm schreiben.

```
1025 DIM PM$(1)
```

Nun ist es wichtig, daß wir das Programm mit LIST speichern, den Programmbereich mit NEW löschen und das Programm dann mit ENTER wieder laden, damit die Variable PM$ die erste Variable in den Tabellen ist. Dies darf nicht vergessen werden, da das Programm sonst 'abstürzen' kann.

Bevor wir dann die Player definieren, müssen wir den String PM$ an die gewünschte Stelle legen, und Länge und Dimensionierung ändern. Dies machen wir wieder in einem eigenen Unterprogramm (am Anfang die üblichen Bemerkungen).

```
3145 GOSUB 5000

5000 REM ***********************************************
5010 REM *       P/M BEWEGUNG VORBEREITEN              *
5020 REM ***********************************************
```

Zunächst müssen wieder die aktuellen Werte der beiden Zeiger VVTP und STARP errechnet werden.

```
5030 VVTP=PEEK(134)+256*PEEK(135)
5040 STARP=PEEK(140)+256*PEEK(141)
```

Für den Zeigereintrag in die Variablenwerttabelle benötigt man nun die Differenz zwischen der gewünschten Anfangsadresse des Strings und dem Anfang der Stringtabelle.

```
5050 DIFF=PMSTRT-STARP
```

Dieser Wert muß nun in die Tabelle eingetragen werden.

```
5060 POKE VVTP+2,DIFF-256*INT(DIFF/256)
5070 POKE VVTP+3,INT(DIFF/256)
```

Nun ändern wir Dimensionierung und Länge des Strings auf die gewünschten Werte.

```
5080 POKE VVTP+4,0
5090 POKE VVTP+5,4
5100 POKE VVTP+6,0
5110 POKE VVTP+7,4
```

Die Länge des Strings wird gleich auf 1024 gesetzt, damit wir jetzt den gesamten Player-Missile-Bereich bequem löschen können.

```
5120 PM$(1,1)=CHR$(0):PM$(2)=PM$
```

Nun können wir die einzelnen Bewegungsphasen des Männchens in Strings speichern. Dazu dienen die Strings P1$, P2$ und P3$.

```
5130 DIM P1$(11),P2$(11),P3$(11)
```

Die einzelnen Strings können natürlich nach Belieben geändert werden, oder Sie erzeugen noch mehr Bewegungsphasen, um auch noch die waagerechte Bewegung des Männchens zu verfeinern. Hier die von uns gewählte Darstellung.

```
5140 RESTORE 5180
5150 FOR SCHLEIFE=1 TO 11
5160 READ BYTE:P1$(SCHLEIFE)=CHR$(BYTE)
5170 NEXT SCHLEIFE
5180 DATA 0,0,8,28,8,28,28,16,16,0,0
5190 FOR SCHLEIFE=1 TO 11
5200 READ BYTE:P2$(SCHLEIFE)=CHR$(BYTE)
5210 NEXT SCHLEIFE
5220 DATA 0,0,8,28,8,28,28,4,4,0,0
5230 FOR SCHLEIFE=1 TO 11
5240 READ BYTE:P3$(SCHLEIFE)=CHR$(BYTE)
5250 NEXT SCHLEIFE
```

```
5260 DATA 0,0,8,28,8,28,28,8,24,0,0
5270 RETURN
```

Jetzt können wir uns daran machen, die Bewegungsroutinen zu ändern.

Mit einer FOR...NEXT-Schleife wird dabei das Männchen schrittweise nach oben, bzw. nach unten verschoben, wobei immer zwischen den beiden Bitmustern für die Bewegung hin und her gesprungen werden muß. Das Programm dazu sieht wie folgt aus:

```
4040 FOR SCHLEIFE=1 TO 8 STEP 4
4050 PM$(40+4*Y+SCHLEIFE)=P1$
4060 FOR SCHLEIFE=1 TO 9:NEXT SCHLEIFE
4070 PM$(42+4*Y+SCHLEIFE)=P2$
4080 FOR SCHLEIFE=1 TO 9:NEXT SCHLEIFE
4090 NEXT SCHLEIFE
4100 Y=Y+2
4110 PM$(39+4*Y)=P3$
4120 RETURN
4140 FOR SCHLEIFE=1 TO 8 STEP 4
4150 PM$(40+4*Y-SCHLEIFE)=P1$
4160 FOR SCHLEIFE=1 TO 9:NEXT SCHLEIFE
4170 PM$(38+4*Y-SCHLEIFE)=P2$
4180 FOR SCHLEIFE=1 TO 9:NEXT SCHLEIFE
4190 NEXT SCHLEIFE
4200 Y=Y-2
4210 PM$(39+4*Y)=P3$
4220 RETURN
```

Auf diese Art und Weise haben wir nun ein sehr einfaches und elegantes Klettern des Männchens erreicht.

Um zu vermeiden, daß die Unterroutine ab Zeile 5000 bei einem weiteren Spiel noch einmal aufgerufen wird, muß man das Unterprogramm ab Zeile 7000 noch etwas abändern.

```
7071 SCORE=0
7072 ZBON=1
7073 TIME=500
7074 X=15:Y=2
7080 GOSUB 3150
```

4.6 RENUMBER / Fehlermeldungen

An dieser Stelle wollen wir noch etwas näher auf die Statement Table (also den 'eigenlichen Programmteil')

eingehen. Um zu sehen, wie Basic ein Programm in verkürzter (tokenized) Form speichert, betrachten wir folgendes kleine Beispiel:

```
1000 VAR=1:PRINT VAR
```

Dieses kleine Programm sollte man in den Rechner eingeben, nachdem man vorher den ganzen Programmbereich mit NEW gelöscht hat. Läßt man sich dann die ersten 19 Byte der Statement Table ausgeben, z.B. mit

```
FOR I=0 TO 18:PRINT PEEK(I);" ";:NEXT I
```

so erhält man folgendes:

```
3 232 19 15 54 128 45 14 64 1 0 0 0 0 20 19 32 128 22
```

Um den Sinn dieser Zahlen zu verstehen, gehen wir sie der Reihe nach durch.

3
232 — Die ersten beiden Byte geben immer die Zeilennummer an. Wie üblich verwendet auch das Basic hier die Form LSB, MSB.

19 — Die Zahl hinter der Zeilennummer gibt jeweils die Länge der Zeile an. Um also den Beginn der nächsten Zeile zu finden, muß man also lediglich diesen Wert zur Adresse dieser Zeile addieren.

15 — Da eine Basic-Zeile (meistens) in mehrere Anweisungen zerfällt, die durch Doppelpunkte getrennt sind, steht am Anfang jeder dieser Anweisungen beim wievielten Byte ab Beginn der Zeile die nächste Anweisung beginnt. Zählt man vom Beginn der Zeile 15 Byte ab, so gelangt man zur 19, die wieder ein Index auf den Beginn der nächsten Anweisung ist.

54 — Unter allen Basic-Befehlen gibt es einige, die am Anfang einer Zeile stehen dürfen. Dies sind die sogenannten Statements. Umgekehrt beginnt jede Anweisung mit einem Statement. In Ausdrücken wie VAR=1 wird deshalb ein LET eingefügt. Die 54 gibt also ein 'unsichtbares' LET an.

128 — Zahlen, die größer als 127 sind, sind ein Zeiger auf eine Variable, insofern sie nicht in einer Konstanten auftreten. Von dieser Zahl ist 128 abzuziehen, um die Nummer der Variablen zu erhalten.

45 — Zeichen bzw. Befehle, die zwei Ausdrücke miteinander verbinden, nennt man Operatoren. Die 45 steht

für den Operator '='.

14 Die 14 kennzeichnet jeweils den Beginn einer numerischen Konstante.

64
1
0
0
0
0

Diese 6 Byte geben den Wert der Konstanten im sogenannten BCD-Format an.

20 Steht für den Doppelpunkt, und gibt somit das Ende dieser Anweisung an.

19 Wie oben die 15 ein Index auf den Beginn der nächsten Anweisung. Da die PRINT-Anweisung die letzte in dieser Zeile ist, zeigt dieser Wert auf den Beginn der nächsten Zeile. Hier steht also derselbe Wert wie im dritten Byte.

32 Hier beginnt eine neue Anweisung, d.h. die 32 steht für ein Statement (PRINT).

128 Nummer der Variablen VAR (wie oben)

22 Marke für Zeilenende

Folgende Liste gibt die Tokens für alle Basic-Statements, -Operatoren und -Funktionen an.

Token dez.	Statement	Token dez.	Operator	Token dez.	Funktion
0	REM	14	(num.	61	STR$
1	DATA		Konstante)	62	CHR$
2	INPUT	15	(String	63	USR
3	COLOR		Konstante)	64	ASC
4	LIST	16	"	65	VAL
5	ENTER	17		66	LEN
6	LET	18	,	67	ADR
7	IF	19	$	68	ATN
8	FOR	20	:	69	COS
9	NEXT	21	;	70	PEEK
10	GOTO	22	(Zeilenende)	71	SIN
11	GO TO	23	GOTO	72	RND
12	GOSUB		(bei ON)	73	FRE
13	TRAP	24	GOSUB	74	EXP

Token dez.	Statement	Token dez.	Operator	Token dez.	Funktion
14	BYE		(bei ON)	75	LOG
15	CONT	25	TO	76	CLOG
16	COM		(bei FOR)	77	SQR
17	CLOSE	26	STEP	78	SGN
18	CLR	27	THEN	79	ABS
19	DEG	28	#	80	INT
20	DIM	29	<= (num.)	81	PADDLE
21	END	30	<> (num.)	82	STICK
22	NEW	31	>= (num.)	83	PTRIG
23	OPEN	32	<	84	STRIG
24	CLOAD	33	>		
25	SAVE	34	= (num.)		
26	STATUS	35	^		
27	NOTE	36	*		
28	POINT	37	+		
29	XIO	38	-		
30	ON	39	/		
31	POKE	40	NOT		
32	PRINT	41	OR		
33	RAD	42	AND		
34	READ	43	(		
35	RESTORE	44	)		
36	RETURN	45	= (num.)		
37	RUN	46	= (String)		
38	STOP	47	<= (String)		
39	POP	48	<>		
40	?	49	>=		
41	GET	50	<		
42	PUT	51	>		
43	GRAPHICS	52	=		
44	PLOT	53	+ (Vorzchn.)		
45	POSITION	54	- (Vorzchn.)		
46	DOS	55	((String)		
47	DRAWTO	56	((Array)		
48	SETCOLOR	57	((bei DIM Array)		
49	LOCATE	58	((bei Funktion)		
50	SOUND	59	((bei DIM Strings)		
51	LPRINT	60	, (bei Arrays)		
52	CSAVE				
53	CLOAD				
54	(LET)				
55	ERROR- (in fehlerhafter Zeile)				

Daß in dieser Tabelle einige Zahlen sowohl für Statements als auch für Operatoren vorkommen, stört nicht, da die Statements ausschließlich am Beginn einer Anweisung stehen.

Stringkonstanten werden in folgender Art und Weise gehandhabt: Das erste Byte nach der 15 (Token für eine Stringkonstante) gibt die Länge der Konstanten an, und dann folgen die einzelnen Bytes dieses Strings.

RENUMBER

Als erstes Hilfsprogramm wollen wir ein RENUMBER-Programm schreiben, also eine Routine, die die Zeilen umnummeriert. Dabei ändern wir jedoch nur die Zeilennummern selbst und nicht die Werte bei GOTO, GOSUB usw., da diese Werte im BCD-Format vorliegen, so daß das Ändern erstens sehr kompliziert und zweitens sehr zeitraubend wäre. Wer sich in Assemblerprogrammierung auskennt, kann sich natürlich selbst so ein Hilfsprogramm schreiben.

Wie immer in diesem Kapitel zuerst das Listing:

```
32600 GRAPHICS 0
32601 PRINT "↓↓   STARTZEILE ";:INPUT XSTART
32602 XEND=32599
32603 TRAP 32607
32604 PRINT "   ENDZEILE  ";:INPUT XEND
32605 IF XEND>=32600 THEN 32604
32606 TRAP 40000
32607 PRINT "↓   ZEILENNUMMER ";:INPUT XNUM
32608 PRINT "   ZEILENABSTAND ";:INPUT XABST
32609 XZGR=PEEK(136)+256*PEEK(137)
32610 XZNUM=PEEK(XZGR)+256*PEEK(XZGR+1)
32611 IF XZNUM>XEND THEN END
32612 IF XZNUM>=XSTART THEN 32615
32613 XZGR=XZGR+PEEK(XZGR+2)
32614 GOTO 32610
32615 PRINT XZNUM,XNUM
32616 POKE XZGR,XNUM-256*INT(XNUM/256)
32617 POKE XZGR+1,INT(XNUM/256)
32618 XNUM=XNUM+XABST
32619 XZGR=XZGR+PEEK(XZGR+2)
32620 XZNUM=PEEK(XZGR)+256*PEEK(XZGR+1)
32621 IF XZNUM<=XEND THEN 32615
```

In Zeile 32601 fragen wir zunächst nach der Zeilennummer, ab der umnummeriert werden soll. Die Zeilennummer bis zu der umnummeriert wird setzen wir zunächst auf 32599, d.h. auf die größtmögliche Zeilennummer bis zu der umnummeriert werden kann. Dies hat den Vorteil, daß man für die Endzeile nicht immer die höchste Zeilennummer eingeben muß,

die im Programm vorkommt. Damit aber kein Fehler auftritt, falls man auf die Frage ENDZEILE ? nur RETURN drückt, muß man vorher einen TRAP-Befehl setzen. Die Endzeilennummer darf nicht größer als 32599 sein, da wir sonst den Programmteil umnummerieren würden, der gerade vom Basic bearbeitet wird (Abfrage in 32605). In den Zeilen 32607 und 32608 wird dann nach der ersten neuen Zeilennummer und dem neuen Zeilenabstand gefragt.

Die Variable XZGR dient uns als Zeiger innerhalb der Statement Table. Um die Adresse der Startzeile zu finden, muß man die Statement Table von vorne weg durchsuchen, bis die richtige Zeilennummer gefunden ist. Daher wird der Zeiger XZGR zunächst auf den Wert in SMTTAB gestellt (Zeile 32609). Die erste Zeilennummer findet man dann -wie oben erklärt-, in der Adresse auf die XZGR zeigt und in der darauffolgenden. Dieser Wert wird in die Variable XZNUM übernommen (Zeile 32610). Ist dieser Wert größer, bzw. genauso groß wie die eingegebene Zeilennummer, dann hat man die Startzeile (bzw. die erste Zeile, die einen Wert größer als die Startzeile hat) gefunden (Abfrage in Zeile 32612). Im anderen Fall wird die Länge der Zeile zu dem Wert in XZGR dazuaddiert und der ganze Vorgang wiederholt (Zeilen 32613 und 32614).

In Zeile 32615 werden die alte und die neue Zeilennummer ausgegeben. Dies soll eine kleine Hilfestellung sein, wenn anschließend die Sprungadressen geändert werden müssen. Wer einen Drucker besitzt, kann hier den Befehl LPRINT verwenden. Damit wird dann das Umnummerieren relativ einfach.

In den Zeilen 32616 und 32617 wird dann die neue Zeilennummer in das Programm geschrieben. In den folgenden Zeilen wird die neue Zeilennummer um den ensprechenden Wert erhöht, XZGR auf die nächste Zeile im Programm gestellt, und die alte Zeilennummer in XZNUM übernommen. Ist die Endzeile erreicht, so ist man fertig (Abfrage in 32615).

Fehlermeldungen

Das zweite Programm ist ein 'Fehlermelder', der nicht nur deutsche Fehlermeldungen im Klartext ausgibt, sondern auch alle in der Fehlerzeile vorkommenden Variablen mit ihren Werten auflistet. Das Programm ist zwar nicht übermäßig schnell, aber auch dieses Programm kann man in ein Maschinenprogramm umschreiben. Wir wollen hier nur die Funktionsweise von solchen Programmen aufzeigen.

```
0 TRAP 32600
32600 GRAPHICS 0
32601 XNUM=PEEK(186)+256*PEEK(187)
32602 XERR=PEEK(195)
32603 XZGR=PEEK(190)+256*PEEK(191)
32604 PRINT "↓↓↖*** FEHLER ****↓"
32605 IF XERR<22 THEN GOSUB 32690+XERR:GOTO 32607
32606 GOSUB 32590+XERR
32607 LIST XNUM
32609 XVNTP=PEEK(130)+256*PEEK(131)
32610 XVVTP=PEEK(134)+256*PEEK(135)
32617 REM --------------- SUCHE VARIABLE -----------------
32618 XZGR=XZGR+3
32619 XZGR=XZGR+1
32620 XSTMT=PEEK(XZGR)
32621 IF XSTMT<2 OR XSTMT>54 THEN END
32622 XZGR=XZGR+1
32623 XOFV=PEEK(XZGR)
32624 IF XOFV=22 THEN END
32625 IF XOFV=20 THEN XZGR=XZGR+1:GOTO 32619
32626 IF XOFV=14 THEN XZGR=XZGR+6:GOTO 32622
32627 IF XOFV=15 THEN XZGR=XZGR+PEEK(XZGR+1)+1:GOTO 32622
32628 IF XOFV<128 THEN 32622
32629 XVNUM=XOFV-128
32630 REM ------------- SUCHE VARIABLENNAME -------------
32631 XVZGR=XVNTP-1
32632 XCNT=0
32633 IF XVNUM=XCNT THEN 32638
32634 XVZGR=XVZGR+1
32635 IF PEEK(XVZGR)<128 THEN 32634
32636 XCNT=XCNT+1
32637 GOTO 32633
32638 XVZGR=XVZGR+1
32639 XCH=PEEK(XVZGR)
32640 IF XCH<128 THEN PRINT CHR$(XCH);:GOTO 32638
32641 PRINT CHR$(XCH-128);"=";
32642 REM ---------------- GIB WERT AUS ----------------
32643 XVZGR=XVVTP+8*XVNUM
32644 IF PEEK(XVZGR)<>0 THEN PRINT :GOTO 32622
32645 XEXP=PEEK(XVZGR+2)
32646 IF XEXP>127 THEN PRINT "-";:XEXP=XEXP-128
32647 XNUM=0
32648 FOR XSCHL=XVZGR+3 TO XVZGR+7
32649 XNUM=100*XNUM+PEEK(XSCHL)-6*INT(PEEK(XSCHL)/16)
32650 NEXT XSCHL
32651 XEXP=XEXP-68
32652 IF XEXP=0 THEN 32657
32653 FOR XVZGR=XEXP TO SGN(XEXP) STEP -SGN(XEXP)
32654 XNUM=(XEXP>0)*XNUM*100+(XEXP<0)*XNUM/100
32656 NEXT XVZGR
32657 PRINT XNUM
```

```
32658 GOTO 32622
32692 PRINT "ZU WENIG SPEICHERPLATZ":RETURN
32693 PRINT "KEIN INTEGER-WERT":RETURN
32694 PRINT "ZU VIELE VARIABLEN":RETURN
32695 PRINT "STRINGLAENGE FALSCH":RETURN
32696 PRINT "ZU WENIG DATEN FUER READ":RETURN
32697 PRINT "ZEILENNUMMER GROESSER 32767":RETURN
32698 PRINT "BEI INPUT KEINE ZAHL EINGEGEBEN":RETURN
32699 PRINT "DIMENSIONIERUNGSFEHLER":RETURN
32700 PRINT "AUSDRUCK ZU KOMPLEX":RETURN
32701 PRINT "FLIESSKOMMAFEHLER":RETURN
32702 PRINT "ZEILE EXISTIRT NICHT":RETURN
32703 PRINT "NEXT OHNE FOR":RETURN
32704 PRINT "ZEILE ZU LANG":RETURN
32705 PRINT "GOSUB- ODER FOR-ZEILE GELOESCHT":RETURN
32706 PRINT "RETURN OHNE GOSUB":RETURN
32707 PRINT :RETURN
32708 PRINT "UNGUELTIGES STRING-ZEICHEN":RETURN
32709 PRINT "PROGRAMM ZU LANG":RETURN
32710 PRINT "FALSCHE KANALNUMMER":RETURN
32711 PRINT "FALSCHES DATENFORMAT":RETURN
32712 PRINT "DISKETTE VOLL":RETURN
32713 PRINT "UNBEHEBBARER SYSTEMFEHLER":RETURN
32714 PRINT "DISKETTE FEHLERHAFT":RETURN
32715 PRINT "UNGUELTIGER DATEINAME":RETURN
32716 PRINT "FEHLER BEI POINT":RETURN
32717 PRINT "DATEI SCHREIBGESCHUETZT":RETURN
32718 PRINT "ABBRUCH DURCH BREAK":RETURN
32719 PRINT "KANAL IST SCHON OFFEN":RETURN
32720 PRINT "GERAET EXISTIERT NICHT":RETURN
32721 PRINT "GERAET SCHREIBT NUR":RETURN
32722 PRINT "UNGUELTIGES KOMMANDO":RETURN
32723 PRINT "KANAL NICHT GEOEFFNET":RETURN
32724 PRINT "UNGUELTIGE KANALNUMMER":RETURN
32725 PRINT "GERAET LIEST NUR":RETURN
32726 PRINT "DATEIENDE ERREICHT":RETURN
32727 PRINT "DATENSATZ ABGESCHNITTEN":RETURN
32728 PRINT "GERAET ANTWORTET NICHT":RETURN
32729 PRINT "UEBERTRAGUNGSFEHLER":RETURN
32730 PRINT "UEBERTRAGUNGSFEHLER":RETURN
32731 PRINT "FALSCHE CURSORKOORDINATEN":RETURN
32732 PRINT "UEBERTRAGUNGSFEHLER":RETURN
32733 PRINT "UEBERTRAGUNGSFEHLER":RETURN
32734 PRINT "DISKETTE SCHREIBGESCHUETZT":RETURN
32735 PRINT "FEHLERHAFT ABGESPEICHERT":RETURN
32736 PRINT "UNBEKANNTES KOMMANDO":RETURN
32737 PRINT "ZU WENIG SPEICHER FUER GRAFIK":RETURN
32750 PRINT "FALSCHE LAUFWERKNUMMER":RETURN
32751 PRINT "ZU VIELE OFFENE DATEIEN":RETURN
32752 PRINT "DISKETTE VOLL":RETURN
32753 PRINT "UNBEHEBBARER SYSTEMFEHLER":RETURN
```

```
32754 PRINT "DISKETTE FEHLERHAFT":RETURN
32755 PRINT "UNGUELTIGER DATEINAME":RETURN
32756 PRINT "FEHLER BEI POINT":RETURN
32757 PRINT "DATEI SCHREIBGESCHUETZT":RETURN
32758 PRINT "UNGUELTIGES KOMMANDO":RETURN
32759 PRINT "INHALTSVERZEICHNIS VOLL":RETURN
32760 PRINT "DATEI NICHT GEFUNDEN":RETURN
32761 PRINT "POINT NICHT ERLAUBT":RETURN
```

ERRMEL **32600 - 32761**

Variablen:

Name	Bedeutung
XNUM	Zeilennummer der Zeile, in der der Fehler aufgetreten ist / Wert der gesuchten Variablen
XERR	Fehlernummer
XZGR	Zeiger in die Statement Table
XVNTP	Zeiger auf die Variablennamentabelle
XVVTP	Zeiger auf die Variablenwerttabelle
XSTMT	aktuelles Statement
XOFV	Token für einen Operator, eine Funktion oder eine Variable
XVNUM	Nummer der gefundenen Variable (mit 0 beginnend)
XVZGR	Zeiger in die Variablennamentabelle, bzw. in die Variablenwerttabelle
XCNT	Zähler für Variablennamen
XCH	Zeichen im gesuchten Variablennamen
XEXP	Exponent bei der BCD-Darstellung
XSCHL	Schleifenparameter

Die Zeile 0 dient dazu den Fehlermelder zu initialisieren, da bei Autreten eines Fehlers immer nach Zeile 32600 verzweigt werden soll. Das hat zur Folge, daß dieses Programm nicht unbedingt angesprungen wird, falls das eigene Programm mit TRAP-Befehlen arbeitet. Dies ist jedoch nicht weiter schlimm, da der TRAP-Modus ohnehin nur immer für kurze Zeit eingeschaltet werden sollte.

In den Basicvariablen STOPLN und ERRSAV findet man die Zeilennummer der fehlerhaften Zeile und die Fehlernummer. Diese beiden Werte werden in die Variablen XNUM und XERR übernommen (Zeilen 32601 und 32602). Der Zeiger XZGR zum Durchsuchen der Statement Table nach der Fehlerzeile wird zunächst auf den Wert der Basicvariablen SAVCUR (Adresse 190,191) gestellt. Dieses Register beinhaltet immer die Adresse der Zeile, die als letztes mit einem Sprungbefehl verlassen wurde. Ein solcher Sprungbefehl ist im Prinzip auch der TRAP, so daß man dieses Register an dieser Stelle verwenden und sich so langes Durchsuchen der Statement Table ersparen kann. Im Allgemeinen ist jedoch von der Verwendung dieses Registers abzuraten.

In den Zeilen 32604 bis 32607 wird dann die Zeile, in der der Fehler aufgetreten ist, gelistet, und die Fehlermeldung ausgegeben.

In den Zeilen 32609 und 32610 werden die aktuellen Werte der Basic-Zeiger in die Register XVNTP und XVVTP übernommen.

Jetzt muß die aktuelle Programmzeile nach Variablen durchsucht werden. Wie wir schon gesehen haben, erkennt man eine Variable immer daran, daß ihr Token einen Wert größer 127 hat. Leider ist aber nicht jeder Wert in der Statement Table, der größer als 127 ist, mit einer Variablen identisch, da es sich hierbei auch um den Teil einer numerischen, bzw. Stringkonstanten handeln kann. Wir kommen also nicht umhin, die aktuelle Programmzeile schrittweise durchzugehen. Dazu erhöhen wir zunächst den Zeiger um 3. Damit deutet er auf das vierte Byte, das den Abstand zum nächsten Statement angibt (Zeile 32618). In der folgenden Zeile wird dieser Wert noch einmal um eins erhöht, so daß der Zeiger nun auf das erste Statement zeigt. Der Wert (Token) dieses Statements wird in die Variable XSTMT übernommen. Handelt es sich hierbei um einen REM- oder DATA-Befehl oder einen SYNTAX ERROR, so können wir das Programm beenden, da in diesen Fällen keine Variablen ausgegeben werden müssen. Ansonsten wird der Zeiger erneut um eins erhöht, so daß er nun auf einen Operator, auf eine Funktion oder auf eine Variable deutet. Dieses Token schreiben wir in die Variable XOFV. Handelt es sich hierbei um das Zeilenende-Zeichen, so sind wir fertig. Im Falle eines Doppelpunktes erhöhen wir den Zeiger um eins, um das Byte, das den Abstand zum nächsten Statement angibt zu umgehen. An allen anderen Funktionen sind nur noch die von besonderem Interesse, die den Beginn einer Konstanten angeben. In diesen Fällen erhöhen wir den Zeiger um 6, falls es sich um eine numerische Konstante handelt, bzw. um die Länge der Stringkonstanten+1, falls

es sich um eine solche handelt. Das '+1' wegen dem zusätzlichen Byte für die Längenangabe in der Statement Table. Ist der Wert in XOFV größer als 127, so kann man nun sicher sein eine Variable vor sich zu haben, da ja alle Konstanten übersprungen werden. Handelt es sich um keine Variable wird solange weitergesucht, bis entweder eine gefunden wird, oder das Programm auf eine der oben genannten Arten verlassen wird.

In Zeile 32629 wird die Nummer der Variablen gespeichert. In dem Programmteil von 32630 bis Zeile 32641 wird dann die Variablennamentabelle durchlaufen, bis der Name der aktuellen Variablen gefunden wird. Dabei zählen wir in der Variablen XCNT die Variablennamen mit, und geben den Namen aus, falls dieser Wert mit dem Wert in XVNUM übereinstimmt.

Im Programmstück von Zeile 32642 bis Zeile 32658 wird dann der Wert der Variablen ausgegeben, insofern es sich um einen Skalar handelt. Da für jede Variable in der Variablenwerttabelle 8 Byte freigehalten werden, muß man also den Wert in XVNUM mit 8 multiplizieren und zu dem Wert in XVVTP addieren, um die Adresse des richtigen Eintrages in der Variablenwerttabelle zu erhalten. Wie wir gesehen haben gibt das erste Byte den Typ dieser Variablen an. Liegt ein Skalar vor, so steht hier eine Null. In allen anderen Fällen suchen wir in der Statement Table nach der nächsten Variablen (Zeile 32644). Im Falle eines Skalars geben wir den BCD-Wert der Variablen aus. Wir wollen hier nicht näher auf diese Dekodierung eingehen -wer will kann sich diesen Programmteil intensiver ansehen.

Damit wären wir am Ende dieses Buches angelangt und möchten Sie noch auf die beiden Folgebände aufmerksam machen, in denen wir uns näher mit der Hardware (256 Farben gleichzeitig auf dem Bildschirm, 'weiches' Scrollen, etc.) und mit dem Betriebssystem (wie funktioniert das DOS, GRAPHICS-Befehle vom Assembler, etc.) beschäftigen.

Anhang

Anhang 1 : Gesamtlisting Bergwerk

```
1000 REM ************************************************** 48
1010 REM *                          SPIEL                 *
1020 REM ************************************************** 48
1025 DIM PM$(1)
1030 POKE 106,PEEK(740)-4
1040 GOSUB 2000
1050 ZSATZ=256*PEEK(106)
1060 GOSUB 2500
1070 PMSTRT=ZSATZ+512
1080 GOSUB 3000
1090 GOSUB 3500
1100 REM ------------- BEGINN HAUPTPROGRAMM -------------
1110 ST=STICK(0)
1120 LOCATE X,Y,L
1130 IF L=42 OR L=134 THEN GOSUB 4230:GOTO 1180
1140 GEHT=USR(ADR(TEST$),ST,L-34)
1150 IF GEHT<15 THEN 1170
1160 FOR SCHLEIFE=1 TO 20:NEXT SCHLEIFE:GOTO 1340
1170 GOSUB 4000
1180 IF X<>14 OR Y<>2 OR MANN=0 THEN 1230
1190 SCORE=SCORE+VPOS+50
1200 ZBON=1.8*ZBON
1210 MANN=0
1220 GOSUB 3500
1230 IF PEEK(53260)=0 THEN 1270
1240 POKE 53278,1
1250 MANN=1
1260 GOSUB 4500
1270 IF X<>2 OR Y<>2 OR ZBON<10 THEN 1340
1280 ZBON=INT(ZBON)
1290 IF ZBON<500 THEN TIME=TIME+ZBON:GOTO 1310
1300 TIME=TIME+500
1310 SCORE=SCORE+TIME
1320 ZBON=1
1330 POKE 657,20:PRINT "      ";
1340 POKE 657,8:PRINT SCORE;
1350 POKE 657,20:PRINT INT(ZBON);
1360 TIME=TIME-1
1370 POKE 657,32:PRINT TIME;" ";
1380 IF TIME=0 THEN 7000
1390 GOTO 1100
2000 REM ************************************************** 48
2010 REM *                NEUE DISPLAY LIST               *
```

```
2020 REM ****************************************************
2030 GRAPHICS 8
2040 POKE 559,0
2050 RAMEND=256*PEEK(106)
2060 BILDSCH=RAMEND-2600
2070 DLIST=BILDSCH-81
2080 RESTORE 2120
2090 FOR SCHLEIFE=DLIST TO DLIST+3
2100 READ BYTE:POKE SCHLEIFE,BYTE
2110 NEXT SCHLEIFE
2120 DATA 112,112,112,79
2130 POKE DLIST+4,BILDSCH-256*INT(BILDSCH/256)
2140 POKE DLIST+5,INT(BILDSCH/256)
2150 FOR SCHLEIFE=DLIST+6 TO DLIST+60
2160 POKE SCHLEIFE,15
2170 NEXT SCHLEIFE
2180 FOR SCHLEIFE=DLIST+61 TO DLIST+76
2190 POKE SCHLEIFE,6
2200 NEXT SCHLEIFE
2210 POKE DLIST+77,2
2220 POKE DLIST+78,65
2230 POKE DLIST+79,DLIST-256*INT(DLIST/256)
2240 POKE DLIST+80,INT(DLIST/256)
2250 POKE 560,PEEK(DLIST+79)
2260 POKE 561,PEEK(DLIST+80)
2270 POKE 559,34
2280 BILD=BILDSCH+2240
2290 POKE 87,1
2300 POKE 88,BILD-256*INT(BILD/256)
2310 POKE 89,INT(BILD/256)
2320 TEXT=BILD+320
2330 POKE 660,TEXT-256*INT(TEXT/256)
2340 POKE 661,INT(TEXT/256)
2350 POKE 752,1
2360 RETURN
2500 REM ****************************************************
2510 REM *                SPIELFELD ZEICHNEN                *
2520 REM ****************************************************
2530 REM
2540 REM ----------------- ZEICHENSATZ COPIEREN ---------
2550 FOR SCHLEIFE=0 TO 511
2560 POKE ZSATZ+SCHLEIFE,PEEK(57344+SCHLEIFE)
2570 NEXT SCHLEIFE
2580 REM ------------------NEUE ZEICHEN ------------------
2590 RESTORE 2650
2600 FOR SCHLEIFE=3 TO 14
2610 FOR ZEILE=0 TO 7
2620 READ BYTE:POKE ZSATZ+8*SCHLEIFE+ZEILE,BYTE
2630 NEXT ZEILE
2640 NEXT SCHLEIFE
2650 DATA 0,0,0,0,0,195,195,255
```

```
2660 DATA 195,195,195,255,195,195,195,255
2670 DATA 0,0,0,0,0,0,0,255
2680 DATA 221,217,249,249,249,249,249,255
2690 DATA 255,255,255,217,217,217,217,217
2700 DATA 48,60,60,120,124,252,62,126
2710 DATA 128,128,128,128,128,128,128,255
2720 DATA 24,24,24,24,24,56,60,255
2730 DATA 126,127,55,126,94,124,60,28
2740 DATA 24,24,60,60,126,126,255,255
2750 DATA 1,1,1,1,1,1,1,255
2760 DATA 12,28,28,24,56,60,116,124
2770 POKE 756,ZSATZ/256
2780 REM -------------------- BILD ZEICHNEN -------------
2790 SETCOLOR 0,2,6
2800 SETCOLOR 1,11,10
2810 SETCOLOR 2,0,0
2820 SETCOLOR 3,0,8
2830 PRINT #6;"    ó   £ó£ £ , ó £ ";:REM ó = CTRL-N
2840 PRINT #6;"  ý ä   äää ä ý ä ä ";:REM £ = CTRL-H
2850 PRINT #6;")éò%*%%é****%*%ò%*é*-";:REM ý = INV CTRL-G
2860 PRINT #6;" ç      ç          ç  ";:REM ä = CTRL-K
2870 PRINT #6;")ç%%é%%ç%%é%%%%%éç%-";:REM é = INV CTRL-C
2880 PRINT #6;"    ç      ç      ç   ";:REM ç = INV CTRL-D
2890 PRINT #6;")%é%ç%-)%%ç%é%-)ç%%-";:REM ò = INF CTRL-F
2900 PRINT #6;"  ç          ç"
2910 PRINT #6;")éç%%%-)é%%%ç%%%é%%-";
2920 PRINT #6;" ç       ç        ç"
2930 PRINT #6;")ç%é-)%%ç%%%%%-)çé%-";
2940 PRINT #6;"   ç              ç"
2950 PRINT #6;")é%ç-)é%-)%é%%é-)çé-";
2960 PRINT #6;" ç    ç    ç  ç   ç"
2970 PRINT #6;")ç%-)%ç%%%%ç-)ç%%%ç-";
2975 REM DIE NAECHSTEN DIE DREI WOERTER
2976 REM I N V E R S  SCHREIBEN
2980 PRINT "↖SCORE        ZBON        ZEIT";
2990 RETURN
3000 REM ***************************************************45
3010 REM *              ANFANGSBEDINGUNGEN                 *
3020 REM ***************************************************45
3030 DIM TEST$(16)
3040 RESTORE 3080
3050 FOR SCHLEIFE=1 TO 16
3060 READ BYTE:TEST$(SCHLEIFE)=CHR$(BYTE)
3070 NEXT SCHLEIFE
3080 DATA 104,104,133,213,104,133,203,104,104
3090 DATA 5,203,41,15,133,212,96
3100 SCORE=0
3110 ZBON=1
3120 TIME=500
3130 X=15:Y=2
3140 REM ---------------- PLAYER DEFINIEREN -------------
```

```
3145 GOSUB 5000
3150 RESTORE 3190
3160 FOR SCHLEIFE=1 TO 7
3170 READ BYTE:POKE PMSTRT+47+SCHLEIFE,BYTE
3180 NEXT SCHLEIFE
3190 DATA 8,28,8,28,28,8,24
3200 POKE 704,14
3205 POKE 623,8
3210 POKE 53248,168
3220 POKE 53276,16
3230 POKE 54279,ZSATZ/256
3240 POKE 53277,2
3250 POKE 559,46
3260 RETURN
3500 REM ***************************************************** 48
3510 REM *                    VERLETZTER                     *
3520 REM ***************************************************** 48
3530 VPOS=8*INT(RND(0)*SCORE/100)
3540 IF VPOS>40 THEN VPOS=40
3550 RESTORE 3610
3560 POKE 53249,48+INT(RND(0)*152)
3570 POKE 705,14
3580 FOR SCHLEIFE=1 TO 3
3590 READ BYTE:POKE PMSTRT+188+VPOS+SCHLEIFE,BYTE
3600 NEXT SCHLEIFE
3610 DATA 90,127,26
3620 RETURN
4000 REM ***************************************************** 48
4010 REM *                BEWEGUNGSROUTINEN                  *
4020 REM ***************************************************** 48
4030 IF ST<>13 THEN 4130
4040 FOR SCHLEIFE=1 TO 8 STEP 4
4050 PM$(40+4*Y+SCHLEIFE)=P1$
4060 FOR SCHLEIFE1=1 TO 9:NEXT SCHLEIFE1
4070 PM$(42+4*Y+SCHLEIFE)=P2$
4080 FOR SCHLEIFE1=1 TO 9:NEXT SCHLEIFE1
4090 NEXT SCHLEIFE
4100 Y=Y+2
4110 PM$(39+4*Y)=P3$
4120 RETURN
4130 IF ST<>14 THEN 4230
4140 FOR SCHLEIFE=1 TO 8 STEP 4
4150 PM$(40+4*Y-SCHLEIFE)=P1$
4160 FOR SCHLEIFE1=1 TO 9:NEXT SCHLEIFE1
4170 PM$(38+4*Y-SCHLEIFE)=P2$
4180 FOR SCHLEIFE1=1 TO 9:NEXT SCHLEIFE1
4190 NEXT SCHLEIFE
4200 Y=Y-2
4210 PM$(39+4*Y)=P3$
4220 RETURN
4230 IF ST<>7 THEN 4300
```

```
4240 XX=48+8*X
4250 FOR SCHLEIFE=2 TO 8 STEP 2
4260 POKE 53248,XX+SCHLEIFE
4270 NEXT SCHLEIFE
4280 X=X+1
4290 RETURN
4300 IF ST<>11 THEN RETURN
4310 XX=48+8*X
4320 FOR SCHLEIFE=2 TO 8 STEP 2
4330 POKE 53248,48+8*X-SCHLEIFE
4340 NEXT SCHLEIFE
4350 X=X-1
4360 RETURN
4500 REM ************************************************** 48
4510 REM *                  LOESCHE VERLETZTEN            *
4520 REM ************************************************** 48
4530 FOR SCHLEIFE=1 TO 7
4540 POKE PMSTRT+188+VPOS+SCHLEIFE,0
4550 NEXT SCHLEIFE
4560 RETURN
5000 REM ************************************************** 48
5010 REM *        PM BEWEGUNG VORBEREITEN                 *
5020 REM ************************************************** 48
5030 VVTP=PEEK(134)+256*PEEK(135)
5040 STARP=PEEK(140)+256*PEEK(141)
5050 DIFF=PMSTRT-STARP
5060 POKE VVTP+2,DIFF-256*INT(DIFF/256)
5070 POKE VVTP+3,INT(DIFF/256)
5080 POKE VVTP+4,0
5090 POKE VVTP+5,2
5100 POKE VVTP+6,0
5110 POKE VVTP+7,2
5120 PM$(1,1)=CHR$(0):PM$(2)=PM$
5130 DIM P1$(11),P2$(11),P3$(11)
5140 RESTORE 5180
5150 FOR SCHLEIFE=1 TO 11
5160 READ BYTE:P1$(SCHLEIFE)=CHR$(BYTE)
5170 NEXT SCHLEIFE
5180 DATA 0,0,8,28,8,28,28,16,16,0,0
5190 FOR SCHLEIFE=1 TO 11
5200 READ BYTE:P2$(SCHLEIFE)=CHR$(BYTE)
5210 NEXT SCHLEIFE
5220 DATA 0,0,8,28,8,28,28,4,4,0,0
5230 FOR SCHLEIFE=1 TO 11
5240 READ BYTE:P3$(SCHLEIFE)=CHR$(BYTE)
5250 NEXT SCHLEIFE
5260 DATA 0,0,8,28,8,28,28,8,24,0,0
5270 RETURN
7000 REM ************************************************** 48
7010 REM *                    NEUES SPIEL                 *
7020 REM **************************************************
```

```
7030 GOSUB 4500
7040 FOR SCHLEIFE=1 TO 7
7050 POKE PMSTRT+39+4*Y+SCHLEIFE,0
7060 NEXT SCHLEIFE
7070 IF STRIG(0)=1 THEN 7070
7071 SCORE=0
7072 ZBON=1
7073 TIME=500
7074 X=15:Y=2
7080 GOSUB 3150
7090 GOSUB 2980
7100 GOTO 1090
```

Anhang 2 : Variablenübersicht BERGWERK

BERGWERK **1000 - 7100**

Variablen:

Name	Bedeutung
BILD	Startadresse des GRAPHICS-1-Bildteils
BYTE	Byte aus DATA-Zeilen
BILDSCH	Startadresse des Bildschirm-RAM = Startadresse des GRAPHICS-8-Bildteils
DIFF	Zeiger für PM$ in der Variablenwerttabelle
DLIST	Beginn der Display List
GEHT	Gibt an, in welche Richtungen gegangen werden darf
L	Zeichen, auf dem das Männchen gerade steht
MANN	=1 : Verletzter aufgenommen
PMSTRT	Startadresse P/M-Bereich
RAMEND	Wert des Zeigers RAMTOP (*256)
SCHLEIFE	Allgemeiner Schleifenparameter
SCORE	Aktueller Punktestand
ST	Joystickwert
STARP	Zeiger auf den Basic-String-Bereich
TEXT	Startadresse des Textfensters
TIME	Spielzeit
VPOS	Vertikale Position des Verletzten
VVTP	Zeiger auf die Variablenwerttabelle
X	aktuelle X-Koordinate des Männchens (im GRAPHICS-1-Koordinatensystem)
XX	X-Koordinate des Männchens (absoluter P/M Wert
Y	aktuelle Y-Koordinate des Männchens (im GRAPHICS-1-Koordinatensystem)
ZBON	Zeitbonus
ZEILE	Schleifenparameter
ZSATZ	Adresse des neuen Zeichensatzes

Strings:

Name	Länge	Bedeutung
PM$	(1024)	P/M-Bereich
P1$	(11)	Bitmuster Männchen
P2$	(11)	Bitmuster Männchen
P3$	(11)	Bitmuster Männchen
TEST$	(16)	Maschinenprogramm zum Testen der möglichen Bewegungsrichtungen

Unterprogrammaufrufe :

in	nach	Zweck
1040	2000	Neue Display List schreiben
1060	2500	Spielfeld zeichnen
1080	3000	Anfangsbedingungen herstellen
1090	3500	Verletzten zeichnen
1130	4230	nur senkrechte Bewegung
1170	4000	Männchen bewegen
1220	3500	Verletzten neu zeichnen
1260	4500	Verletzten löschen
3145	5000	P/M-Bewegung vorbereiten
7030	4500	Verletzten löschen
7080	3100	Spielstandsanzeigen auf Anfangswerte setzen
7090	2980	Spielstandsanzeigen auf den Bildschirm schreiben

Anhang 3 : Parametertypen

Tabelle der Parametertypen:

A - Ausgabeparameter von diesem Unterprogramm
E - Eingabeparameter für dieses Unterprogramm
G - Globale Variable
H - Hilfsvariable
P - Aufrufparameter an Unterprogramm
R - Rückgabeparameter von Unterprogramm
T - Transienter Parameter (ist in einem Unterprogramm gleichzeitig Eingabeparameter (E) und Aufrufparameter (P) bzw. A und R

Globale Variablen gibt es zwar bei Basic nicht, da Basic keine block-orientierte Sprache wie z.B. PL/1 ist, aber in diesem Buch ist dieser Begriff für Variablen verwendet worden, die man in anderen Sprachen global definiert hätte.

Da der verwendete Typenraddrucker die Zeichen 'größer als' und 'kleiner als' nicht drucken kann, wurden diese Zeichen wie folgt ersetzt:

NE - Not Equal	/ ungleich
GT - Greater Than	/ größer als
LT - Less Than	/ kleiner als
GE - Greater or Equal	/ größer gleich
LE - Less or Equal	/ kleiner gleich

Anhang 4 : Interner Code und ATASCII-Code der Zeichen

Zeichen	ATASCII Code	interner Code	Tastendruck
	0	64	CTRL ,
	1	65	CTRL A
	2	66	CTRL B
	3	67	CTRL C
	4	68	CTRL D
	5	69	CTRL E
	6	70	CTRL F
	7	71	CTRL G
	8	72	CTRL H
	9	73	CTRL I
	10	74	CTRL J
	11	75	CTRL K
	12	76	CTRL L
	13	77	CTRL M
	14	78	CTRL N
	15	79	CTRL O
	16	80	CTRL P
	17	81	CTRL Q
	18	82	CTRL R

Zeichen	ATASCII Code	interner Code	Tastendruck
✚	19	83	CTRL S
✚	20	84	CTRL T
▬	21	85	CTRL U
▌	22	86	CTRL V
▜	23	87	CTRL W
▟	24	88	CTRL X
▌	25	89	CTRL Y
▙	26	90	CTRL Z
▞	27	91	ESC ESC
✚	28	92	ESC CTRL —
✚	29	93	ESC CTRL =
✚	30	94	ESC CTRL +
✚	31	95	ESC CTRL *
	32	0	
!	33	1	
"	34	2	
#	35	3	
$	36	4	
%	37	5	
&	38	6	
'	39	7	
(	40	8	
)	41	9	
*	42	10	
+	43	11	
,	44	12	

Zeichen	ATASCII Code	interner Code	Tastendruck
—	45	13	
.	46	14	
/	47	15	
0	48	16	
1	49	17	
2	50	18	
3	51	19	
4	52	20	
5	53	21	
6	54	22	
7	55	23	
8	56	24	
9	57	25	
:	58	26	
;	59	27	
<	60	28	
=	61	29	
>	62	30	
?	63	31	
@	64	32	
A	65	33	
B	66	34	
C	67	35	
D	68	36	
E	69	37	
F	70	38	
G	71	39	
H	72	40	
I	73	41	
J	74	42	
K	75	43	
L	76	44	
M	77	45	
N	78	46	

Zeichen	ATASCII Code	interner Code	Tastendruck
O	79	47	
P	80	48	
Q	81	49	
R	82	50	
S	83	51	
T	84	52	
U	85	53	
V	86	54	
W	87	55	
X	88	56	
Y	89	57	
Z	90	58	
[	91	59	
\	92	60	
]	93	61	
^	94	62	
—	95	63	
♣	96	96	CTRL .
a	97	97	
b	98	98	
c	99	99	
d	100	100	
e	101	101	
f	102	102	
g	103	103	
h	104	104	
i	105	105	
j	106	106	
k	107	107	
l	108	108	
m	109	109	
n	110	110	

Zeichen	ATASCII Code	interner Code	Tastendruck
o	111	111	
p	112	112	
q	113	113	
r	114	114	
s	115	115	
t	116	116	
u	117	117	
v	118	118	
w	119	119	
x	120	120	
y	121	121	
z	122	122	
♣	123	123	CTRL :
|	124	124	
↖	125	125	ESC CTRL
◀	126	126	
▶	127	127	

Anhang 5 : Grafikmodes / Textmodes

Basic Mode	SETCOLOR Wert	COLOR Wert	Beschreibung
Mode 0 und alle Textfenster	0	bestimmt das zu plottende Zeichen	-
	1		Helligkeit der Zeichen
	2		Hintergrund
	3		-
	4		Umrandung
Modes 1 und 2	0	bestimmt Zeichen und Farbe	Zeichen
	1		Zeichen
	2		Zeichen
	3		Zeichen
	4		Hintergrund/Umrandung
Modes 3,5,7 und 15	0	1	Bildpunkt
	1	2	Bildpunkt
	2	3	Bildpunkt
	3	-	-
	4	0	Hintergrund/Umrandung
Modes 4,6 und 14	0	1	Bildpunkt
	1	-	-
	2	-	-
	3	-	-
	4	0	Hintergrund/Umrandung
Mode 8	0	-	-
	1	1	Bildpunkt (Helligkeit)
	2	0	Hintergrund
	3	-	-
	4	-	Umrandung
Mode 12 und 13	0	bestimmt Zeichen und Farbe	Zeichen
	1		Zeichen
	2		Zeichen
	3		Zeichen (normal)
	4		Zeichen (invers)
Mode 9,10,11 siehe Kap. 2.2			

Basic Mode	ANTIC Mode	Art der Darst.	RAM pro Zeile	Höhe pro Pixel in scan lines	Farben a=absol. I=Index
0	2	Text	40	8	1.5 (I)
1	6	Text	20	8	5 (I)
2	7	Text	20	16	5 (I)
3	8	Grafik	10	8	4 (I)
4	9	Grafik	10	4	2 (I)
5	10	Grafik	20	4	4 (I)
6	11	Grafik	20	2	2 (I)
7	13	Grafik	40	2	4 (I)
8	15	Grafik	40	1	1.5 (I)
9	-	Grafik	40	1	1*16(a)
10	-	Grafik	40	1	9 (I)
11	-	Grafik	40	1	16*1 (a)
12	4	Text	40	8	4-5 (I)
13	5	Text	40	16	4-5 (I)
14	12	Grafik	20	1	2 (I)
15	14	Grafik	40	1	4 (I)
-	3	Text	40	10	1.5 (I)

Ein 'I' bedeutet dabei, daß der COLOR-Wert als Index auf ein Farbregister ist und ein 'a' bedeutet, daß der COLOR-Wert die absolute Farbe angibt.

Anhang 6 : GRAPHICS 12 für 400/800 Besitzer

Den Befehl GRAPHICS 12, der von den älteren Atari-Computern nicht verstanden wird, kann mit folgendem Hilfsprogramm simuliert werden.

```
10000 REM GR.12 SIMULATOR
10010 GRAPHICS 0
10020 DLIST=PEEK(560)+256*PEEK(561)
10030 POKE DLIST+3,68
10040 FOR I=DLIST+6 TO DLIST+24
10050 POKE I,4
10060 NEXT I
10070 POKE 703,4
```

Anhang 7 : Betriebssystem- und Basicvariablen

In der folgenden Tabelle sind die Variablen nach ihren Adressen geordnet angegeben, außerdem wird ihre Funktion kurz erklärt.

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
77	$4D	ATRACT	attract mode timer

Wird in der Zeit, wo keine Taste gedrückt wird hochgezählt, um bei Erreichen von 128 das zufällige Wechseln der Farben einzuschalten. Um dies bei Programmen, die über Joystick oder Paddle laufen, zu vermeiden, ist hier ab und zu der Wert 0 hineinzuschreiben.

82	$52	LMARGN	left margin

Linke Begrenzung im Grafikmode 0 und in den Textfenstern.

83	$53	RMARGN	right margin

Rechte Begrenzung im Grafikmode 0 und in den Texfenstern.

84	$54	ROWCRS	row cursor

Aktuelle Y-Koordinate des Cursors im Grafikmode 0 und in allen anderen Modes, aber nicht in den Textfenstern. Die oberste Zeile hat dabei den Wert 0 (s.TXTROW 656).

85,86	$55,56	COLCRS	column cursor

Aktuelle X-Koordinate des Cursors im Graifkmode 0 und in allen anderen Modes, aber nicht in den Textfenstern. Der Wert 0 entspricht der linken Spalte (s. TXTCOL 657,658).

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
87	$57	DINDEX	display index

aktueller Grafikmode (Betriebssystem)

88,89	$58,59	SAVMSC	save memory scan counter

Bildschirmspeicherplatz der linken oberen Ecke (Betriebssystem)

106	$6A	RAMTOP	RAM top

Adresse des höchsten RAM-Byte+1 (11SB). Von dieser Adresse ab wird der Bildschirm und die Display List gelegt. (In Richtung der niedrigeren Adressen)

128,129	$80,81	LOMEM	low memory

Adresse des niedrigsten vom Betriebsystem nicht verwendeten Byte; Beginn Basic (Basic)

130,131	$82,82	VNTP	variable name table pointer

Zeiger auf den Beginn der Tabelle mit den Namen der Variablen (Basic).

132,133	$83,84	VNTD	variable name table dummy

Zeiger auf das Ende der Variablennamentabelle (Basic).

134,135	$85,86	VVTP	variable value table pointer

Zeiger auf den Beginn der Tabelle mit den Werten der Variablen (Basic).

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
136,137	$88,89	STMTAB	statement table

Zeiger auf die Statement Table, also auf das eigentliche Programm in übersetzter (tokenized) Form (Basic).

138,139	$8A,8B	STMCUR	current statement

Zeiger auf das gerade bearbeitete Statement (Basic).

140,141	$8C,8D	STARP	string/array table pointer

Zeiger auf den Bereich, wo alle Strings und dimensionierte Variablen gespeichert sind (Basic).

142,143	$8E,8F	RUNSTK	run time stack

Zeiger auf den Beginn des Kellerspeichers, den Basic für GOSUB's und FOR...NEXT-Schleifen benötigt (Basic).

144,145	$90/91	MEMTOP	memory top

Adresse des ersten freien Bytes hinter dem vom Basicprogramm (genauer: dem Kellerspeicher) belegten Speicherplatz (Basic).

186,187	$BA,BB	STOPLN	stop line

Nummer der Zeile in der ein Fehler aufgetreten ist (Basic).

195	$C3	ERRSAVE	error save

Fehlernummer (Basic).

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens

212,213 $D4,D5

Dieser Wert wird von der USR-Funktion zurückgegeben. (Basic)

559 $22F SDMCTL shadow direct memory access (DMA) control

Verantwortlich für folgende Optionen:

Option	Dezimal	Bit
Bild	2	1
Missile	4	2
Player	8	3
Ein-Zeilen-Auflösung	16	4
Display List lesen	32	5

Da das Lesen der Display List immer erforderlich ist, sind die Werte darüber in geigneter Weise zu 32 zu addieren (Hardware).

560,561 $230,231 SDLSTL shadow display list start low

Anfangsadresse der Display List (Hardware).

623 $26F GPRIOR global priority

Verantwortlich für die GTIA Modes und einige Player-Missile Optionen , siehe Kapitel 3 (Hardware).

656 $290 TXTROW text window row

Y-Koordinate des Cursors in den Textfenstern (s. ROWCRS)

657,658 $291,292 TXTCOL text window column

X-Koordinate des Cursors in den Textfenstern (s. COLCRS)

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
660,661	$294,295	TXTMSC	text window memory scan counter

Adresse der linken oberen Ecke im Textfenster (s. SAVMSC 88,89)

665	$299		

Interner Code des Zeichens unter dem Cursor im Textfenster

704	$2C0	PCOLR0	player 0 color
:	:	:	:
712	$2C8	COLOR4	color playfield 4

Farbregister

740	$2E4	RAMSIZ	RAM size

Adresse des höchsten RAM Byte (MSB)

741,742	$2E5,2E6	HIMEM	high memory

Bis zu dieser Adresse kann das Basicprogramm gehen. Wer befürchtet, daß seine selbsterstellte Display List durch Anwachsen des Basic Programms oder des Kellerspeichers zerstört wird, sollte in HIMEM den Wert von SDLSTL (560, 561) -1 setzen.

752	$2F0	CRSWH	cursor inhibit

Jeder Wert außer der Null schaltet den Cursor aus. Die Änderung wird jedoch erst bei der nächsten Ausgabe sichtbar.

755	$2F3	CHACT	character control

Probieren Sie: Poke 755,6; der normale Wert ist 2.

756	$2F4	CHBAS	character base

Startadresse des Zeichensatzes (Hardware).

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
764	$2FC	CH	character

Ist keine Taste gedrückt, steht hier 255, ansonsten der Tastaturwert des gedrückten Zeichens. Dieser Wert bleibt solange in CH stehen, bis ein Eingabebefehl erfolgt (INPUT, GET)

766	$2FE	DSPFLG	display flag

Schreibt man hier einen anderen Wert als 0 hinein, so werden alle Steuerzeichen nicht als Funktion, sondern als Zeichen ausgegeben.

842	$34A		

POKE 842,13 versetzt den Rechner in den automatischen RETURN-Mode, d.h. beim Anhalten des Programm wird ewiges Drücken der RETURN-Taste simuliert. Darum sollte man vorher an geeignete Stelle ein CONT oder ein POKE 842,12 auf den Bildschirm schreiben.

53248(s)	$D000	HPOSP0	horizontal position player 0
:	:	:	:
53255(s)	$D007	HPOSM3	horizontal position missile 3

Horizontale Positionen der Player-Missile (siehe Kapitel 3)

53256(s)	$D00B	SIZEP0	size player 0
:	:	:	
53259(s)	$D00B	SIZEP3	size player 3

Horizontale Größe der Player (siehe Kapitel 3)

53260(s)	$D00C	SIZEM	size of (all) missiles

Horizontale Größe der Missiles (siehe Kapitel 3)

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens
53248(l)	$D000	MOPF	missile 0 to playfield collision
:	:	:	:
53263(l)	$D00F	P3PL	player 3 to player collision

Kollisionsregister (siehe Kapitel 3)

53276(s) $D01C VDELAY vertical delay

Gibt die Möglichkeit einzelne Objekte in Zwei-Zeilen-Auflösung um eine scan line nach unten zu verschieben (siehe Kapitel 3)

53277(s) $D01D GRACTL graphics control

Der Wert 1 erlaubt Darstellung von Missile, der Wert 2 erlaubt Player und 3 erlaubt beides.

53278(s) $D01E HITCLR hit clear

Wird dieses Register mit POKE angesprochen, so werden alle Kollisionsregister gelöscht.

53279(l) $D01F CONSOL console

die niedrigsten drei BIT geben die Zustände der OPTION- sowie SELECT- und START-Taste in folgender Weise wieder:

Taste	Bits	Dezimal
OPTION	011	3
SELECT	101	5
START	110	6

Da jede Taste ein Bit auf Null setzt, kann man damit auch abfragen, ob mehrere Tasten gleichzeitig gedrückt sind.
z.B. OPTION + SELECT ergibt: 001; dezimal 1

Adresse dezimal	Adresse hexadez.	Name	Bedeutung des Namens

53769(l) $D209 KBCODE keyboard code

Enthält den Tastencode der gerade gedrückten Taste. Steht hier 17, so ist die HELP-Taste gedrückt. Die HELP-Taste ist also nichts anderes als eine normale Taste, kann jedoch mit INPUT und GET nicht abgefragt werden. Abfrage:
IF PEEK (53279)=17 AND PEEK (53775)=251 THEN 'gedrückt'

53770(l) $D20A RANDOM random

Enthält eine Zufallszahl zwischen 0 und 255

54279(s) $D407 PMPASE player missile base

Startadresse des Player-Missile Bereichs (s. Kapitel 3)

Die folgende Tabelle enthält nun die Register alphanumerisch geordnet.

Name	Adresse dez.	Adresse hex
ATRACT	77	$4D
CH	764	$2FC
CHACT	755	$2F3
CHBAS	756	$2F4
COLCRS	85,86	$55,56
COLOR0	708	$2C4
COLOR1	709	$2C5
COLOR2	710	$2C6
COLOR3	711	$2C7
COLOR4	712	$2C8
CONSOL	53279	$D01F
CRSINH	752	$2F0
DINDEX	87	$57
DSPFLG	766	$2FE
ERRSAVE	195	$C3
GPRIOR	623	$26F
GRACTL	53277	$D01D
HIMEM	741,742	$2E5,2E6
HITCLR	53278	$D01E
HPOSM0	53252	$D004
HPOSM1	53253	$D005
HPOSM2	53254	$D006

Name	Adresse dez.	Adresse hex
HPOSM3	53255	$D007
HPOSP0	53248	$D000
HPOSP1	53249	$D001
HPOSP2	53250	$D002
HPOSP3	53251	$D003
KBCODE	53769	$D209
LMARGN	82	$52
LOMEM	128,129	$80,81
MEMTOP	144,145	$90,91
M0PF	53248	$D000
M1PF	53249	$D001
M2PF	53250	$D002
M3PF	53251	$D003
M0PL	53256	$D008
M1PL	53257	$D009
M2PL	53258	$D00A
M3PL	53259	$D00B
PCOLR0	704	$2C0
PCOLR1	705	$2C1
PCOLR2	706	$2C2
PCOLR3	707	$2C3
PMBASE	54279	$D407
P0PF	53252	$D004
P1PF	53253	$D005
P2PF	53254	$D006
P3PF	53255	$D007
P0PL	53260	$D00C
P1PL	53261	$D00D
P2PL	53262	$D00E
P3PL	53263	$D00F
RANDOM	53770	$D20A
RAMSIZ	740	$2E4
RAMTOP	106	$6A
RMARGN	83	$53
ROWCRS	84	$54
RUNSTK	142,143	$8E,8F
SAVMSC	88,89	$58,59
SDLSTL	560,561	$320,231
SDMCTL	559	$22F
SIZEM	53260	$D00C
SIZEP0	53256	$D008
SIZEP1	53257	$D009
SIZEP2	53258	$D00A
SIZEP3	53259	$D00B
STARP	140,141	$8C,8D
STMCUR	138,139	$8A,8B
STMTAB	136,137	$88,89
STOPLN	186,187	$BA,BB

Name	Adresse dez.	Adresse hex
TXTCOL	657,658	$291,292
TXTMSC	660,661	$294,295
TXTROW	656	$290
VDELAY	53276	$D01C
VNTD	132,133	$84,85
VNTP	130,131	$82,83
VVTP	134,135	$86,87

Für die drei nachfolgend aufgeführten Register existieren keine gesonderten Namen

USR-Wert	212,213	$D4,D5
Zeichen im Textfenster	665	$299
Selbst-RETURN-Mode	842	$34A

Anhang 8 : Assemblerlisting zum Kopieren des Zeichensatzes (ZNEU)

```
                1000 ; ****************************************
                1010 ; *  ROUTINE ZUM ZEICHENSATZ KOPIEREN    *
                1020 ; *  NUR VON BASIC AUS AUFRUFBAR         *
                1030 ; ****************************************
                1040 ;
                1050 ;
                1060 ;              VARIABLEN
                1070 ;
00CE            1080 ZGR1    =    206        (ZEIGER IN DEN
                1090 ;                       ORIGINALZ.SATZ)
00D0            1100 ZGR2    =    208        (ZEIGER IN DEN
                1110 ;                       NEUEN ZSATZ)
                1120 ;
                1130 ;
0000            1140         *=   1536
                1150 ;
0600 68         1160         PLA             (ZIEHE ANZAHL DER
                1170 ;                       UEBERGEBENEN VA-
                1180 ;                       RIABLEN VOM STACK)
0601 68         1190         PLA             (ZIEHE
0602 85D1       1200         STA  ZGR2+1     MSB
0604 68         1210         PLA             UND
0605 85D0       1220         STA  ZGR2       LSB VOM STACK
                1230 ;                       UND SPEICHERE)
0607 85CE       1240         STA  ZGR1       (WIE OBEN IST DAS
                1250 ;                       LSB 0)
0609 A9E0       1260         LDA  #224       (DAS MSB
060B 85CF       1270         STA  ZGR1+1     IST 224)
060D A000       1280         LDY  #0         (ERSTER INDEX)
060F B1CE       1290 LOOP    LDA  (ZGR1),Y   (KOPIERE
0611 91D0       1300         STA  (ZGR2),Y   EIN ZEICHEN)
0613 C8         1310         INY             (ERHOEHE INDEX)
0614 D0F9       1320         BNE  LOOP       (WENN 256 NOCH
                1330 ;                       NICHT ERREICHT
                1340 ;                       IST MACHE WEITER)
0616 E6D1       1350         INC  ZGR2+1     (ERHOEHE DIE
0618 E6CF       1360         INC  ZGR1+1     BEIDEN MSB'S)
061A A5CF       1370         LDA  207        (WENN DAS ENDE
061C C9E4       1380         CMP  #228       NOCH NICHT ER-
061E D0EF       1390         BNE  LOOP       REICHT IST, MACH
                1400 ;                       WEITER
0620 60         1410         RTS             SONST FERTIG)
```

Anhang 9 : Buchstaben mit Unterlängen

Dieses Assemblerlisting dient zum Kopieren des Zeichensatzes, wobei die Buchstaben um eine Scan-Line nach unten verschoben werden.

```
                1000 ; ****************************************
                1010 ; *  ROUTINE ZUM ZEICHENSATZ KOPIEREN    *
                1020 ; *  MIT VERSCHIEBEN FUER ANTIC-MODE 3   *
                1030 ; *  NUR VON BASIC AUS AUFRUFBAR         *
                1040 ; ****************************************
                1050 ;
                1060 ;
                1070 ;              VARIABLEN
                1080 ;
00CE            1090 ZGR1    =    206        (ZEIGER IN DEN
                1100 ;                       ORIGINALZ.SATZ)
00D0            1110 ZGR2    =    208        (ZEIGER IN DEN
                1120 ;                       NEUEN ZSATZ)
                1130 ;
                1140 ;
0000            1150         *=   1536
                1160 ;
0600 68         1170         PLA             (ZIEHE ANZAHL DER
                1180 ;                       UEBERGEBENEN VA-
                1190 ;                       RIABLEN VOM STACK)
0601 68         1200         PLA             (ZIEHE
0602 85D1       1210         STA  ZGR2+1     MSB
0604 68         1220         PLA             UND
0605 85D0       1230         STA  ZGR2       LSB VOM STACK
                1240 ;                       UND SPEICHERE)
0607 85CE       1250         STA  ZGR1       (WIE OBEN IST DAS
                1260 ;                       LSB 0)
0609 A9E0       1270         LDA  #224       (DAS MSB
060B 85CF       1280         STA  ZGR1+1     IST 224)
060D A000       1290 LOOP1   LDY  #0         (ERSTE ZEILE
                1300 ;                       DER KODIERUNG)
060F 98         1310         TYA             (SETZE 0 IN DIE
0610 91D0       1320         STA  (ZGR2),Y   ERSTE ZEILE)
0612 B1CE       1330 LOOP2   LDA  (ZGR1),Y   (KOPIERE EIN
0614 C8         1340         INY             BYTE MIT
0615 91D0       1350         STA  (ZGR2),Y   VERSCHIEBEN)
0617 C007       1360         CPY  #7         (WIEDERHOLE, BIS
0619 D0F7       1370         BNE  LOOP2      BUCHSTABE VOLL-
                1380 ;                       STAENDIG KOPIERT
```

```
                1390 ;                          IST)
061B A5D0       1400        LDA   ZGR2          (SETZE LSB
061D 18         1410        CLC                 DER ZEIGER
061E 6908       1420        ADC   #8            AUF
0620 85D0       1430        STA   ZGR2          NAECHSTEN
0622 85CE       1440        STA   ZGR1          BUCHSTABEN
0624 90E7       1450        BCC   LOOP1         UND MACH WEITER)
0626 E6D1       1460        INC   ZGR2+1        (FALLS SEITE UEBER-
0628 A5CF       1470        LDA   ZGR1+1        SCHRITTEN WURDE,
062A 18         1480        CLC                 ERHOEHE
062B 6901       1490        ADC   #1            MSB DER
062D 85CF       1500        STA   ZGR1+1        ZEIGER)
062F C9E4       1510        CMP   #228          (MACH WEITER, BIS
0631 D0DA       1520        BNE   LOOP1         ENDE ERREICHT)
0633 60         1530        RTS                 (FERTIG)
```

Anhang 10 : Inhaltsverzeichnis der Diskette

Name	Länge in Sektoren	von Seite
NIM.BAS	37	15
ANEU.BAS	3	27
ZNEU.BAS	134	31
BEISP.CHN	17	Beispiel für ZNEU
BMU.BAS	14	61
TFENSTER.BAS	2	63
GR11DEMO.BAS	4	64
XIODEMO	3	65
DATAGEN.BAS	8	99
DELLN.LST	6	102
REFLIST.LST	2	107
NAMAEND.LST	6	108
COPY.BAS	6	110
RENUMBER.LST	6	118
ERRMEL.LST	33	120
BERGWERK.BAS	60	127
GR12SIM.BAS	2	144

Aus dem M&T-Buchverlag

W. Pest: **Computerchinesisch für Einsteiger**
Juli 1984, 107 Seiten
Ein praxisnahes Lexikon, das Personal Computer-Benutzern und solchen, die es werden wollen, das Lesen von Fachzeitschriften, Büchern, Bedienungsanleitungen und Datenblättern erleichtert · über 1000 häufig benötigte Fachbegriffe klar und verständlich erläutert · mit zahlreichen Abbildungen.
Best.-Nr. MT 690, DM 28,— (Sfr. 25,90/öS 218,40)

Personal Computer Lexikon
1982, 136 Seiten
Über 1000 Suchbegriffe aus Hard- und Software · deutsch/englisch · ausführlicher Artikel zu jedem Suchbegriff · englisch/deutsch Register im Anhang · der ideale Einstieg ins Homecomputing · das unentbehrliche Nachschlagewerk für den Profi.
Best.-Nr. MT 390, DM 19,80 (Sfr. 18,50/öS 154,40)

M.J.Winter: **Lehrspielzeug Computer: Atari**
Juli 1984, 139 Seiten
Das neue Computer-Kinderbuch für den Atari 400, 800 und 1200 · Spielprogramme und grafische Darstellungen für Kinder ab 8 Jahren · viele Rechenaufgaben für den kleinen Einstein · so macht Lernen Freude!
Best.-Nr. MT 696, DM 24,80 (Sfr. 23,—/öS 193,40)

H. Glicksman/K. Simon: **Spiele für den Atari**
September 1984, 216 Seiten
Eine unterhaltsame Einweisung in die Atari-BASIC-Programmierung anhand von bereits bewährten sowie raffinierten neuen Computerspielen · Wie man ein Programm strukturiert · Einsatz von Unterprogrammen · Tabellenverarbeitung · Bewegte Grafiken · Testen von Programmen · Noch nie hat Home-Computing so viel Spaß gemacht!
Best.-Nr. MT 678, DM 32,— (Sfr. 29,50/öS 249,60)

H. L. Schneider/R. Bichler: **Das Atari-Buch, Bd. 1**
1984, 158 Seiten
Die grundlegenden Programmiermöglichkeiten für Ihren Atari · mit einem Spiel zum Eingewöhnen · Erstellung von Text und Grafik · Player Missiles · Basic-Besonderheiten · ausführliche Assemblerlistings im Anhang.
Best.-Nr. MT 703, DM 32,— (Sfr. 29,50/öS 249,60)

T. Bridge: **Atari-Abenteuerspiele**
1984, 148 Seiten
Alles über die Anfänge der Abenteuerspiele · Textabenteuer mit vielen Rätseln · Schatzsuche · Kampf mit Monstern · Das Auge des Sternenkriegers · mit hilfreichen Anregungen zum Schreiben Ihrer eigenen Spieleprogramme.
Best.-Nr. MT 727, DM 29,80 (Sfr. 27,50/öS 232,40)

J. White
Strategische Computerspiele für Ihren Atari
1984, 148 Seiten
Aufbau eines Spielfeldes · der Bewegungsablauf · Mustereröffnungen · das Endspiel · Dame, Schach, Warp Trog als Beispiele strategischer Spiele · Anleitung zur systematischen Fehlersuche · Grundkenntnisse in Atari-Basic erforderlich.
Best.-Nr. MT 681, DM 32,— (Sfr. 29,50/öS 249,60)

H. Kohl/T. Kahn et al.: **Spiel und Spaß mit dem Atari**
1984, 338 Seiten
Einfache Programme in Basic · wie man ein Spiel entwickelt · Lernstoff trainieren · Zahlen und Logik · Grafik · Farben · Töne und Musik · den Atari-Computer spielend erforschen.
Best.-Nr. MT 672, DM 42,— (Sfr. 38,60/öS 327,60)

H.L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 1
1984, 270 Seiten
Der Commodore 64 und seine Handhabung · Einführung in die Grafik · Balkendiagramme · Einführung in die Spritetechnik · Basic-Erweiterungen in Assembler · Ein Leitfaden für Erstanwender.
Best.-Nr. MT 591 (Buch)
DM 48,— (Sfr. 44,20/öS 374,40)
Best.-Nr. MT 592 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H.L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 2
1984, 181 Seiten
Spiele nicht nur zum Abtippen · Programmlisting · Programmbeschreibung · Variablenübersicht · Programme nach Anleitung frei ergänzbar · das ideale Buch, um Programmieren spielend zu lernen.
Best.-Nr. MT 593 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 594 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 3
1984, 206 Seiten
Alles über Sprites · Wissenswertes über Multi-Color-Grafik · Assembler/Disassembler · jede Menge Basic-Erweiterungen · Umgang mit dem Soundgenerator · ein Leitfaden für Fortgeschrittene.
Best.-Nr. MT 595 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 596 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 4
1984, 261 Seiten
Einführung in Maschinenprogrammierung · Verknüpfung von Maschinenprogrammen mit Basic-Programmen · alles über Assembler/Disassembler · der Leitfaden für Systemprogrammierer.
Best.-Nr. MT 597 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 598 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 5
Juli 1984, 322 Seiten
Ein Leitfaden durch Simon's Basic · ausführliche Besprechung aller Befehle · viele erklärende Beispiele · mit kommentierter Assembler-Listing · das richtige Nachschlagewerk für den geübten Commodore 64-Benutzer.
Best.-Nr. MT 599 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 600 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

Die angegebenen Preise sind Ladenpreise

Sie erhalten M&T-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Str. 2, 8013 Haar, Telefon: 089/4613-220

Aus dem M&T-Buchverlag

S. Urute
Grafik & Musik auf dem Commodore 64
Oktober 1984, 336 Seiten
68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.
Best.-Nr. MT 743, DM 38,— (Sfr. 35,—/öS 296,40)

H. L. Schneider
Commodore 64 Listings — Band 1: Spiele
Oktober 1984, 199 Seiten
Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise
Best.-Nr. MT 748, DM 24,80 (Sfr. 23,—/öS 193,40)

E. H. Carlson: **Lerne Basic auf dem VC-20**
August 1984, 320 Seiten
Das neue Basic-Lehrbuch für den Commodore VC-20 · einfach erklärte Basic-Befehle mit Übungen · viele heiße Actionspiele · nützliche Programmiertricks · mit ausführlichem Begriffslexikon · der Renner für junge Computer-Freaks!
Best.-Nr. MT 691, DM 38,— (Sfr. 35,—/öS 296,40)

P. Rädsch:
Programme und Tips für VC-20
1983, 152 Seiten
Nützliche Hilfsprogramme für die Arbeit mit dem VC-20 · kommerzielle Anwendung in der Textverarbeitung, Fakturierung und Lagerverwaltung · Möglichkeiten hochauflösender Grafik über eine Assemblerroutine · unterhaltsame Spielprogramme.
Best.-Nr. MT 513, DM 38,— (Sfr. 35,—/öS 296,40)

M. Hegenbarth/M. Schäfer: **Das VC-20-Buch**
1983, 351 Seiten
Eine Sammlung gut erklärter Programme · viele Spielebeispiele · einfache kommerzielle Anwendungen
Best.-Nr. MT 516 (Buch)
DM 49,— (Sfr. 45,10/öS382,20)
Best.-Nr. MT 581 (Kassette)
DM 19,90 (Sfr. 19,90/öS 179,10)
Best.-Nr. MT 582 (Diskette)
DM 29,90 (Sfr. 29,90/öS 269,10)

N. Hampshire: **Grafik mit dem VC-20**
1984, 202 Seiten
38 vollständige Programme · zahlreiche grafische Darstellungen · alles über hochauflösende Grafik und Multicolor-Modus · praktische Anwendungen und Simulationen von Kunst über Videospiele, Mathematik, Naturwissenschaften bis hin zum kaufmännischen Bereich.
Best.-Nr. MT 644, DM 32,— (Sfr. 29,50/öS 249,60)

R. Zamora/D. Inman et al.: **Basic mit dem VC-20**
1984, 364 Seiten
Eine schrittweise Einführung in das Gebiet von VC-20-Basic · Geräusch- und Musikerzeugung · Drucken von grafischen Schriftzeichen · Erstellen eines lauffähigen VC-20-Programms · Arbeiten mit Zeichenvariablen, einfachen Federvariablen, READ- und DATA-Befehlen · Zeichentricks.
Best.-Nr. MT 649, DM 38,— (Sfr. 35,—/öS 296,40)

D. Laine
Maschinencode-Programme für den ZX Spectrum
1984, 204 Seiten
Nützliche Maschinencode-Programme mit Ihrem ZX Spectrum · Sortierung von Fließkommazahlen · Übernahme von Parametern direkt von einem Basic-Programm · Flußdiagramme · für Profis und solche, die es werden wollen.
Best.-Nr. MT 702, DM 32,— (Sfr. 29,50/öS 249,60)

M. Gavin
Astronomie-Programme für den ZX-Spectrum
September 1984, 268 Seiten
Eine phantastische Reise in die Welt des Kosmos mit Ihrem ZX-Spectrum: Der Julianische Kalender · Die Mondphasen · Eigene Satelliten starten · Kepler's Umlaufbahnen · Die Umlaufbahn Plutos · Interessant nicht nur für Hobby-Astronome.
Best.-Nr. MT 732, DM 32,80 (Sfr. 27,50/öS 232,40)

T. Bridge/R. Carnell: **ZX-Spectrum Abenteuerspiele**
September 1984, 208 Seiten
Die Entstehungsgeschichte der Abenteuerspiele mit repräsentativen Beispielen für jede »Epoche«. Ein Programm speziell für Ihren ZX-Spectrum: »Das Auge des Sternenkriegers«, ein Grafik-Abenteuerspiel, das Sie in Atem hält!
Best.-Nr. MT 712, DM 29,80 (Sfr. 27,50/öS 232,40)

J. Cassidy/P. Katz et al.: **Im Land der Abenteuer**
1984, 146 Seiten
Eine Hilfestellung für zahlreiche Computerspiele · Tod in der Karibik · Transsylvanien · Unternehmen Asteroid · Das geheimnisvolle Haus · Zauberer und Prinzessin · Das goldene Vlies · Zeitzone · Der dunkle Kristall · mit Lösungen.
Best.-Nr. MT 699, DM 29,80 (Sfr. 27,50/öS 232,40)

J. R. Brown: **Basic für Einsteiger**
1984, 239 Seiten
Ein Arbeitsbuch für den absoluten Anfänger · Basic-Anweisungen Schritt für Schritt erklärt und anhand von einfachen Beispielen erläutert · das beliebte Arbeitsmittel für Lehrkräfte und für den interessierten Computerfan.
Best.-Nr. MT 680, DM 32,— (Sfr. 29,50/öS 249,60)

Ch. Langfelder: **Basic ohne Probleme, Bd. 1**
1983, 226 Seiten
Eine Unterweisung in Basic mit CBM-Rechnern (CMB 8032) · Grundlagen des Betriebssystems · Funktionsweise des Interpreters · mathematische Programme · Verarbeitung von Texten und Zeichen · Glossar der wichtigsten Fachbegriffe.
Best.-Nr. MT 480, DM 36,— (Sfr. 33,10/öS 280,80)

Ch. Langfelder: **Basic ohne Probleme, Bd. 2**
1982, 119 Seiten
Für alle CBM 8032-Rechner · ausgewählte Routinen und Programme · drei allgemeine Routinen · fünf kommerziell-technische Anwendungen · zwei Statistikprogramme · zwei Mathematikprogramme · drei Lehr- und Spielprogramme.
Best.-Nr. MT 490, DM 26,— (Sfr. 24,10/öS 202,80)

Die angegebenen Preise sind Ladenpreise

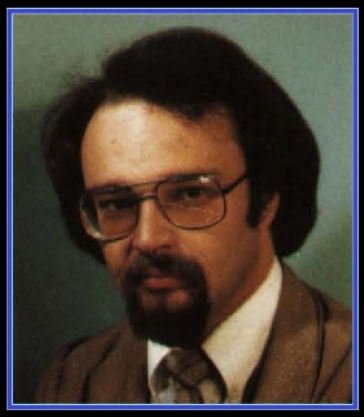
Sie erhalten M&T-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

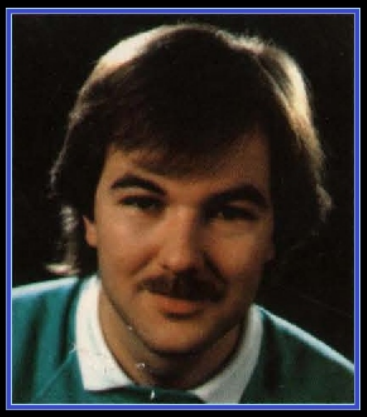
Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Str. 2, 8013 Haar, Telefon: 089/4613-220

Das ATARI-Buch
Band 1: Grundlegende Programmiermöglichkeiten

Mit diesem Buch möchten wir Ihnen die grundlegenden Programmiermöglichkeiten Ihres ATARI-Computers vorstellen, wobei die Kenntnisse des ATARI-Handbuches vorausgesetzt werden. Es wurde versucht, auf eine Art und Weise die doch recht komplizierten Vorgänge in Ihrem Computer dazulegen.
Benutzen Sie das erste Kapitel, um sich etwas mit Ihrem Rechner vertraut zu machen. Hier wird ein einfaches Streichholzspiel auf den Computer übertragen.
ATARI-Computer sind als Spielcomputer konzipiert und dementsprechend wirkt sich auch die Handhabung bei der Programmierarbeit aus. Komplizierte effektvollere Spiele sind relativ leicht zu programmieren. Besonderen Wert wurde auf die verschiedenen Möglichkeiten der Grafik- und Textdarstellung gelegt. Auch eine genaue Beschreibung der Player-Missile wurde in diesen Band einbezogen.

In den weiteren Bänden werden alle Möglichkeiten der Hardware aufgezeigt und es wird näher auf das Betriebssystem eingegangen.





HANS L. SCHNEIDER geboren am 15.10.53 in Köln. Nach dem Abitur 1973 studierte er von 1976 bis 1980 Informatik an der Bundeswehrhochschule in München. Seit 1980 ist Schneider Inhaber und Geschäftsführer eines Software-Hauses, das sich hauptsächlic mit der Erstellung von Individual-Software für Mikrocomputer befaßt.

RUDOLF BICHLER geboren am 31.5.1960 in München. Studiert seit 1981 Mathematik und Informatik an der Universität München. Arbeitet seit längerem als freier Mitarbeiter für ein Softwarehaus, wo er sich hauptsächlich mit der Systemprogrammierung befaßt. ATARI-Computer sind seit einigen Jahren sein Hobby.

Markt&Technik Verlag Aktiengesellschaft
ISBN 3-89090-039-9